File No. S360-36
Form C28-2001-2

TSS

IBM Systems Reference Library

IBM System/360 Time Sharing System

Command System User's Guide

The command system in Time Sharing System/360 gives to the user the facilities he needs for constructing, executing, and debugging his programs; also, he can create, modify, share, and copy data sets; he can move them to or from input/output devices. The user can modify and add to the IBM-supplied command system to meet his specific requirements.

Third Edition (September 1968)

This is a major revision of, and makes obsolete, Form C28-2001-1 and Technical Newsletters N28-3003, N28-3013, and N28-3027.

This publication reflects extensive changes made to the command system. The user has broader and more flexible means of creating and editing VISAM data sets with 16 new text editing commands. Furthermore, the user can alter the IBM-supplied command system to suit his own needs and even write his own commands.

This edition is current with Version 3, Modification 0, and remains in effect for all subsequent versions or modifications of IBM System/360 Time Sharing System unless otherwise indicated. Significant changes or additions to this publication will be provided in new editions or Technical Newsletters. Before using this publication in connection with the operation of IBM systems, refer to the latest edition of IBM System/360 Time Sharing System: Addendum, Form C28-2043, for the editions of publications that are applicable and current.

Requests for copies of IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form is provided at the back of this publication for reader's comments. If the form has been removed, comments may be addressed to IBM Corporation, System/360 Time Sharing System Programming Publications, Department 561, 2651 Strang Boulevard, Yorktown Heights, N. Y. 10598.

The IBM System/360 Time Sharing System, TSS/360, provides specific facilities for each class of system-user. One set of facilities, reserved for the system manager and his administrators, is described in IBM System/360 Time Sharing System: Managers and Administrators Guide, Form C28-2024. Another set, reserved for the system operator, is described in IBM System/360 Time Sharing System: Operator's Guide, Form C28-2033-2.

A third set of facilities, reserved for those who are concerned with using the system is described in this publication. In TSS/360, this type of individual is called a "user," whether he is a system programmer, an applications programmer, or someone who uses the system only on an inquiry basis.

Part I of this publication deals with the basic command system and how to write commands. The following parts describe the specific commands in functional groups. Each command is described under seven points: introductory statement, format illustration, operand descriptions, functional description, programming notes, cautions, and examples.

The appendixes contain reference material such as command formats, printer- and punch-control codes, user-profile tables, and how to submit bulk input to the system.

PREREQUISITE PUBLICATIONS

Effective use of this manual requires an understanding of TSS/360 and the relationships that exist among the individuals who use the system. See IBM System/360 Time Sharing System: Concepts and Facilities, Form C28-2003.

Use of this manual at a terminal requires knowledge of the instructions for operating the IBM 2741 and IBM 1052 terminals in the TSS/360 environment. These are given in IBM System/360 Time Sharing System: Terminal User's Guide, Form C28-2017.

PUBLICATIONS CHECK LIST

IBM System/360 Time Sharing System: System Messages, Form C28-2037

IBM System /360 Time Sharing System: Concepts and Facilities, Form C28-2003

IBM System/360 Time Sharing System: Terminal User's Guide, Form C28-2017

IBM System/360 Time Sharing System: Assembler User Macro Instructions, Form C28-2004

IBM System/360 Time Sharing System: Assembler Programmer's Guide, Form C28-2032

IBM System/360 Time Sharing System: FORTRAN Programmer's Guide, Form C28-2025

IBM System/360 Time Sharing System: Linkage Editor, Form C28-2005

IBM System/360 Time Sharing System: System Programmer's Guide, Form C28-2008

TABLES

The command system is the principal medium of communication between the user and IBM System/360 Time Sharing System (TSS/360). The facilities of the command system permit the user to construct, execute, and debug his programs; to manipulate his data sets; to modify the IBM-supplied command system, and to write his own commands. He can use the command system in two modes of operation: In conversational mode, the user communicates with the system while it is executing operations for him; he remains on-line to the system and maintains a dialogue with it. In nonconversational mode, the user does not maintain a dialogue with the system; the command system serves as a form of job control language, since the user's requests are submitted for execution without his monitoring. Unless otherwise noted in the command descriptions, detailed in this publication, commands may be used in both conversational and nonconversational mode.

COMMANDS

Two types of commands are available to the user: IBM-supplied and user-written. System-supplied commands group functionally in six categories:

Task management
Data management
Program management
Command creation
User profile management
Message handling

The command-creation commands (PROCDEF and BUILTIN) enable the user to augment IBM-supplied command facilities by designing and implementing his own (user-written) commands. PROCDEF creates new commands from a combination of commands; BUILTIN creates new commands from assembler language object code. Once created, user-written commands are specified in the same way as IBM-supplied commands; to the user, there is no difference in the manner of execution.

Note: This publication describes the IBM-supplied command system and the facilities available to the user. Part of these facilities enables the user to alter the command system to suit his needs. The user (and the installation) can define synonyms for IBM-supplied command names, operand names, and variables; he can specify his own default values to overlay system defaults; and he can replace symbols that indicate system requests with other symbols. How this is accomplished is described in Part V, "User Profile Management."

COMMAND FORMAT AND NOTATION

The basic format of a command is:

Operation	Operand
command name	one or more operands, delimited by commas; field may be blank

The operation field contains a command name, such as CANCEL or EXECUTE, that identifies the command and its requested action. The command name may not exceed eight characters or contain an embedded blank. The operand field contains any information required by the command.

While the operation field specifies the action to be performed, the operand field indicates the elements upon which the command is to act. The operand field may be blank or may contain several operands, depending on the requirements of the operation. Multiple operands in an operand field must be separated by commas. Blank and/or tab characters may also be used between operands, but they are ignored by the system. For example, specifying the operands a, b, and c in the operand field of a command as:

```
a,b,c

a  ,b  ,c

or a , (tab) b  ,c
```

will yield identical results when the command is executed. The operand field is separated from the operation field by either a tab character, or one or more blanks.

Command Statement

A command statement is one command or a series of commands that the system recognizes as one SYSIN record. Normally, command statements are written one on a line, unless the last character of the previous line is a continuation character. When a command statement contains more than one command, the commands must be separated by semicolons. The user may add a comment clause, in the form of a quoted string, to a command statement. A comment, which must be separated from the commands in the command statement by a semicolon, has no effect on execution. A comment clause may also begin after a system underscore:

```
___'this is a comment'
```

The three types of command statements are: dynamic, immediate, and conditional. A dynamic statement contains an AT command, which specifies the location where the subsequent commands in the statement will be executed. An immediate statement is any command statement that does not contain an AT command and will be executed when it is entered. A conditional statement is any command statement (dynamic or immediate) that contains an IF command; the part of the statement following IF will be executed only when the condition stipulated by IF is true.

Here are examples of command statements:

1.
```
delete myds; 'erase the catalog entry for myds'; cancel 3912;-
'eliminate print task'
```

2.
```
execute datal; catalog data2,u,u;logoff
```

3.
```
wt dsname=abcd, dsname2=xyz,volume=1233,factor=3,startno=6,-
endno=35,prtsp=edit;'output data set'
```

4.
```
at pgm.a; display x
```

5.
```
if x>0; display x
```

Operand Representation

Command operands are represented in two ways: by position and by keyword. The system can determine the value of a specific operand from either the relative position of that operand within a series of operands, or from a descriptive keyword preceding the operand value.

When positional operands are used, they must be supplied by the user in the same order that was shown in the command format illustration. If a positional operand is omitted, and another positional operand is written following the omitted operand, the comma that would have followed the omitted operand must be retained to indicate the relative position of the operand that is included. For example, positional operands a, b, and c may be written as:

```
a,b,c   a,,c   a,b   a   ,b,c   ,b   ,,c   (blank)
```

Keywords may appear in any order, in the general form: KEYWORD=value, where KEYWORD is the name of the operand, and value is the actual value of the operand. This value is the one that would be specified for the operand in positional representation. Commas are not required to indicate ommitted keyword operands.

Keyword and positional representation of operands may be used simultaneously in the same operand field. Where one operand is expressed in both manners, the last in the series is assumed to be the value. For example, assuming three operands with keyword representations expressed as A=x, B=y, and C=z, the operand field may be represented as:

```
A=x,y,z  A=x,B=y,z  x,C=z,B=y  x,C=Z  x,y,A=d,C=z  B=y
```

Note that in the fifth operand field (x,y,A=d,C=z,), the keyword form of representation for the operand specifying "A=d" contains the value that is assumed, since the last value encountered for each operand, from left to right, is assumed by the system. The examples are a guide; they do not contain all possible permutations for these three operands.

This is the sequence of events that the system goes through to resolve the operands:

1. The list of synonym values is searched for equivalent terms;
2. If synonym values exist, they are inserted; if not,
3. The user-supplied explicit value is used; if it is not given,
4. The list of user default values is searched;
5. If default values exist, they are inserted; if not,
6. Available system default values are inserted; if there are none,
7. The operand is not filled in; it is given a null string value.

Here are examples of the sequence:

1. If a SYNONYN A=B command has been given previously, for this operand list (keywords of A,B,C)

   ```
   ,B=b,c
   ```

   the values generated are

   ```
   A=b,  B=b,  C=c
   ```

2. If a SYNONYM command has not been given, but a DEFAULT A=5 has been entered, the operands above would be resolved as

   A=5, B=b, C=c

3. If neither SYNONYM nor DEFAULT has been entered, the values that would apply are

   B=b, C=c

   A would have a null string value.

FORMAT ILLUSTRATIONS

The notational conventions described in the paragraphs following are used in the command format illustrations to explain to the user how an operand is to be written.

Use of Metasymbols

To facilitate the representation of the statements in the format illustrations, four metasymbols will be used, each in the specified context:

Name	Symbol	Use
braces	{ }	(a) to enclose and thus delimit syntactical units (one or more operands) that may be repeated;
brackets	[ ]	(b) to enclose and thus delimit alternatives, to enclose and thus delimit optional names and/or operands within the appropriate fields.
vertical stroke	|	represents "exclusive or" and separates alternative representations of operands; for example, A|B denotes that for the syntactical unit enclosed within the braces, either alternative A or alternative B, but not both A and B, may be chosen; A|B|C denotes that a choice may be made between alternatives A, B, or C. Alternatives may also be indicated by aligning the choices vertically within the braces: { A / B }
ellipses	...	to indicate that the preceding syntactical unit may be repeated one or more times. Should there be a system limit to the number of repetitions permitted, this will be given in the operand list that follows the format illustration.

Operation Format

To distinguish command names in the format illustrations, upper-case letters are used. The user may enter command names in either upper- or lower-case letters, depending on his mode of input. In folded mode (i.e., upper-case letters and lower-case letters are equivalent), he may use both. Unless specified by the user, this is the normal mode of keyboard input. In full EBCDIC mode (i.e., upper- and lower-case letters

are differentiated by the system), he must use upper-case letters. A detailed description of how to enter commands is given in Part II, under "Communicating with the System."

Operand Format

Within the operand field of the format illustration, the word or phrase that will be used to identify each operand will be written entirely in lower-case letters. For positional operands, only the lower-case word or phrase will appear; for keyword operands, the keyword (to the left of the equal sign) will be in upper-case letters and the keyword descriptor (to the right of the equal sign) will be in lower-case letters.

Note: Unless otherwise noted (in the operand descriptions), all operands shown in keyword format may be specified positionally. The converse is not true; operands shown in positional format must be specified in positional notation.

CODED VALUE: This is a character or string of characters that is to be written exactly as shown in the format illustration. Coded values always appear in format illustrations as numbers or upper-case letters, either to the right of the equal sign or standing alone.

The comma, the period, and the parentheses have special significance in format illustrations. Commas must always be written to separate operands, or to show the omission of positional operands, unless no other operand follows the omission. Parentheses and periods must be written as shown in the illustrations.

Operand Descriptions

Detailed information about writing each operand will be given in a list following every format illustration. Every operand description will conclude with two headings: "Specified as," which describes the valid specifications for the operand, and "System default," which describes the system action if the operand is omitted. System default is not shown if the system's default value is null.

Function and Use

Following the operand description, the command is discussed under "Functional Description," "Programming Notes," and "Cautions."

"Functional Description" describes the action of the system when the command is received. "Programming notes" contains information on how to use the command; if none of this information is pertinent to the particular command, the subheading for these notes will be omitted. "Cautions" are statements of warning to the user about difficulties he may have in using the command. "Cautions" will appear only where applicable.

One or more examples of command usage follow the command description; a brief description of what the user might want to do is followed by his input and the system's response.

These general terms are used in many of the command descriptions (more specific terms, referring to a specific functional group of commands, are defined in the introduction preceding the functional group):

data definition name	The name assigned to the data set definition for a given data set by DDEF. This name consists of one to eight alphameric characters, the first of which must be alphabetic.
data set name	The name used to identify a data set. A data set name consists of one or more simple names, each simple name having one to eight alphameric characters, the first of which must be alphabetic. A period is used as the separator between simple names. Example: GOAT GOAT.WINNER9 GOAT.RALPHR.S66.P1.A The maximum number of characters, including periods, is 35; thus, the maximum number of simple names is 18. Fully qualified data set name: identifies one specific data set; it includes all simple names (i.e., qualifiers or index levels) of that data set name. Partially qualified data set name: identifies two or more data sets by omitting the rightmost simple names of their fully qualified data set names. For example, the partially qualified data set name G0.AB14 identifies data sets G0.AB14.P1 and G0.AB14.P2.
default value	The value that the system or user assigns to an operand that has been omitted.
defined	A data set is defined when its characteristics are described to the system. Every uncataloged data set referred to in a task must be defined within that task; the definition must precede the first reference. A data set may be defined by means of a DDEF command or macro instruction, or by a CDD command that results in execution of a prestored DDEF command.
diagnostic messages	Messages issued by the system to inform the user of an error made in entering a command statement. These messages usually indicate the next action expected.

generation data group

A collection of successive, historically related data sets called generations. The entire group is referred to by a single partially qualified data set name, limited to 26 characters to allow for appending absolute generation numbers.

generation names

Specific generations of a generation data group are referred to by appending an absolute or relative number to the generation data group name.

Absolute generation number: has the form GxxxxVyy, where xxxx is a four-digit decimal generation number and yy is a two-digit decimal version number. Example:

```
HURST.LINER4.TT.G0002V01
HARZ.G0452V23
```

A period must separate the absolute generation number from the generation data group name to which it is appended.

Relative generation number: a plus or minus decimal number. The relative generation number of the most recently cataloged generation is (0); the generation just prior to that is (-1) and the one just prior to (-1) is (-2); a new generation is (+1). Example:

```
GOST.YZ(0)
GOST.FF.PKJ(+1)
```

line

A physical record in a line or region data set. Also, line may refer to a unit of information entered from a terminal, consisting of the string of characters (including blanks) typed in before the RETURN key is pressed.

member name

Identifies a member of a VPAM data set. The member name consists of one to eight alphameric characters, the first of which must be alphabetic. Example:

```
FRH.T4(SWING8)
```

The member name is enclosed in parentheses and immediately follows the VPAM data set name.

volume identifications

The identification assigned to a specific volume. The volume identification consists of one to six alphameric characters.

PART 2: TASK MANAGEMENT

Task management commands allow the user to initiate, terminate, or change the system's operation in his behalf. The term "task" describes any discrete sequence of the system's operations for the user.

In conversational mode, the user's task is the operations performed by the system, including the communication between system and user during any period of continuous operation. In nonconversational mode, although the same definition of task applies, communication is one-way because the user has predefined the operations to be performed by the system. The user may have more than one task in the system at a time, but these tasks are independent of each other.

The task management commands and the system functions they request are shown in Table 1.

Table 1. Task Management Commands and Their Functions (The commands are listed in the same order as the command descriptions that follow in Section 2.)

Command	Function
LOGON	Identify user to system for initiation of his task.
ZLOGON	Perform user-defined function.
TIME	Establish time limit for execution of task.
LOGOFF	Notify system that user wants to terminate his task.
BACK	Shift user's conversational task to nonconversational.
EXECUTE	Initiate previously defined nonconversational task.
SECURE	Identify types of I/O devices needed for private data sets in nonconversational task.
CANCEL	Terminate execution of nonconversational task prior to its normal end.
ABEND	Eliminate current task; start new task.

SECTION 1: COMMUNICATING WITH THE SYSTEM

The user is known to the system by his user identification, which was assigned to him at JOIN time by an installation administrator. All the user's data is stored in the system under his identification. Thus, when the user enters the system, the minimum data required to initiate communication is his user identification.

Conversational Task Initiation

The user initiates his conversational task by turning on his terminal and dialing the system. His conversational task is initiated; the system assumes that the user wants to log on. After issuing the LOGON command and its operands successfully, the user enters his command statements through his terminal keyboard, or card reader, to direct the execution of his task.

SYSIN: This name designates the input stream, which contains the series of command statements that direct the user's task, and may include source language statements and data. In conversational mode, this input stream is entered through the user's terminal. The executable command statements within a conversational SYSIN are recorded only as the printed listing at the terminal; the exceptions are the DATA, MODIFY, and text-editing commands, which are used to build a data set that is recorded within the system.

TIME: As a part of the initialization (LOGON) process, the system automatically invokes the TIME command, establishing a CPU time limit for execution of the user's task. The user may specify a time limit, not exceeding 7-1/2 hours, by issuing the TIME command at any time during his task.

Conversational Task Execution

After the initialization process has been completed, the system asks the user to enter his next command statement (see "Request for Next Command Statement," below) and engages in a conversation with him. The user's part of this dialogue consists of any command and source language statements that he enters during execution of his task, and his replies to the messages issued by the system. The system's part of this dialogue consists of messages to the user, responses to his command statements, and requests for next command statements. The user has control over the length and type of messages he will receive. Details will be presented in Part VII,"Message Handling."

The system issues general information messages and messages informing the user of error conditions.

INFORMATION MESSAGES: These messages prompt the conversational user to supply certain information when a mandatory operand has been omitted, or inform the user of the actions the system has taken in executing a command statement.

DIAGNOSTIC MESSAGES: These messages warn the user of errors that he has made in entering a command name or operands; some messages request the user to correct his errors.

REQUEST FOR NEXT COMMAND STATEMENT: The system informs the user, at his keyboard, that it is ready to accept his next command statement by printing an underscore character (_) in the first character position of a new line. (The same indication is given when the user is entering his command statements through the terminal card reader.)

ENTERING COMMAND STATEMENTS: Command statements may be entered into the system from the user's terminal, the system card reader, or a magnetic input device in which the information is stored in card-image format.

The end of a command statement entered from the terminal keyboard is indicated by pressing the RETURN key. If a command statement requires more than one line, one hyphen must be typed at the end of the line before the RETURN key is pressed; the hyphen signals that the statement is not complete, and will be continued on the next line. Upper- and lower-case notations in this publication are illustrative; command statements may be entered in either form.

Command statements that are entered through the terminal card reader can utilize free-form format (i.e., input is not restricted to particular card fields). The 11-5-9 punch, following the command operands is used to signify end of block (EOB) for command statements. For statements longer than 80 characters, with the terminal EOB switch on, the continuation character may appear in any available column. If the EOB switch is off, the continuation character is not needed unless the statement exceeds 260 characters.

Note: Nonconversational input through a computer center's high-speed card reader does not require the 11-5-9 punch to signify EOB; its inclusion will have no effect. A semicolon is a valid command separator. An EOB is automatically inserted by the card reader at the end of every card. A continuation character must appear (in any column) for command statements that require more than one card.

Caution: In most cases, tab characters are treated as spaces and are valid characters in the command system. However, because of physical limitations in terminal devices, displaying tabs of more than 65 consecutive spaces at the terminal printer might cause the next character to be lost. Furthermore, when two or more consecutive tabs are entered through the terminal card reader, they might not be printed correctly at the terminal printer, even through they will be correctly transmitted to the system.

SYSIN DEVICE AND CHARACTER CONTROL: The user has six commands (listed in Table 2) with which he can select the SYSIN device or the character set he wants to use for communication with the system.

Table 2. Commands for SYSIN Device and Character Set Selection

Command	Function
K	Transfer control to keyboard; if character set used during card reader input was CA, KA will be new mode; if CB was card reader mode, KB will be new mode.
KA	Transfer control to keyboard and use full EBCDIC character set.
KB	Transfer control to keyboard and use folded character set.
C	Transfer control to card reader; if keyboard mode was KA, CA will be new mode; if KB was keyboard mode CB will be new mode.
CA	Transfer control to card reader and convert card input from 1057 card-punch code to EBCDIC.
CB	Transfer control to card reader and convert card input from 029 punch code to EBCDIC.

When the user initiates a conversational task, the system automatically assumes folded mode. The user can type any lower-case letter and ¢, ", and ! which will be converted to their upper-case equivalents and @, #, and $ respectively. The system will accept the full EBCDIC character set when the user enters KA. To initiate card reading, the user enters the C, CA, or CB command at the keyboard and presses the RETURN key. The system will then read all the cards, or will read until the user presses the ATTENTION key, or until a K, KA, or KB command is read. After any of these conditions, the system requests the next input from the keyboard.

For further instructions on SYSIN device selection and character control, refer to Terminal User's Guide.

COMMAND STATEMENT EXECUTION: First, every command statement entered by the user is analyzed to determine if it is valid; then, if it is, the actions requested by the command statement are performed before the user is prompted to enter the next statement. If a command in a command statement is not valid, the system issues a diagnostic message which may request the user's corrections. If the invalid command is canceled, the

rest of the command statement is executed before the system invites the user to enter his correction or next statement. Prompting messages are issued as each command in a statement is analyzed, and the user can supply requested information when the message is issued.

Correctly entered commands will have the same effect whether they are entered in one statement or in individual statements. Example:

```
call abc; print resultds,,,edit; delete gh.k
```

will produce the same result when executed as

```
call abc
print resultds,,,edit
delete gh.k
```

The first example (three commands in one command statement) is more convenient for the user, since he does not have to wait for the execution of each command before he can enter the next command. If, however, the CALL command was entered incorrectly and was canceled by the system, the user would not be able to correct his error in the first case, until the other two commands were executed. In the second case, he would be able to correct his error before PRINT and DELETE were executed.

Conversational Task Interruption

The user can interrupt execution of his conversational task by pressing the ATTENTION key at his terminal. When the attention intervention prevention switch is disabled, the system's normal response depends on when the interruption occurs. This information is described in Table 3.

Table 3. Effect of Attention Interruption

When ATTENTION Key Is Pressed	Effect of Interruption
Command statement being entered (RETURN key not pressed)	Command statement ignored; system responds with logical NOT symbol (¬); user can enter any command.
Command in command statement, or program being executed	In most cases (exceptions are noted in command descriptions), command or program is interrupted and system responds with logical NOT symbol (if I/O operation was interrupted) or with exclamation point. User can enter GO to resume processing, ABEND to cancel statement or program, or any other command; if he enters other commands, he can return to interrupted command or program with GO.
Message being printed at terminal	Remainder of message is not issued; system responds with logical NOT symbol. User can enter REPEAT to have last nonprompting message printed, or he can enter any other command.
Prompting or diagnostic message requests information from user	Execution of command that caused message is cancelled; system issues logical NOT symbol. User can enter any command or return to command following cancelled command by responding to NOT symbol with carriage return. If he issues other commands, user can return to interrupted command stream with GO command.

When the user responds to an ATTENTION interrupt with a GO command or a carriage return (i.e., null command), processing of the interrupted program (i.e., command or user's problem program) is resumed.

When the user responds with the REPEAT command, the last nonprompting message will be displayed at his terminal, and the interrupted processing will be resumed. After an ABEND response, the interrupted program is cancelled, the system is returned to its state immediately after the LOGON process, and the user is again prompted for input of new command statements.

When the user responds with a command other than ABEND, GO, or REPEAT, a special intervention routine saves the interrupted program, and the new command statements are honored. These command statements may also be interrupted. The number of routines a user can interrupt and save by the special intervention routine is limited only by the amount of space in his virtual storage that can be allocated for save areas.

Whenever the user issues a GO command, the system will return control to the most recently interrupted program.

Example: The user wants to display part of his virtual storage after his object program, ABC, is running:

```
User:       abc
            (presses ATTENTION key)
System:     !

User:       display abc2:abc2.(x'200')
System:     (displays requested data field)

Sys,User: _go
After system completes execution of DISPLAY, it types underscore;
GO returns control to ABC.
```

Sometimes, the system may not open the terminal keyboard immediately after the ATTENTION key is pressed; this will happen if the system is either executing a privileged routine or the user has inhibited attention interruptions. Normally, the terminal will be opened upon completion of the privileged routine. However, if the user wants immediate access to the keyboard while he has inhibited attention interruptions, he may press the ATTENTION key a second time; then the terminal will be opened.

Often, the user will want to complete execution of short sequences of code in his own program before allowing an attention interruption. He can set the attention interruption prevention switch (AIPS), which will prevent the first attention interruption that is attempted. When one is attempted, the system tests the AIPS; when that switch is set, the system informs the user that the system is busy in the routine and the interruption is ignored.

However, the system does make a simulated attention interruption entry. Then, when the switch is turned off, the user can execute the AIPS to determine whether an attention interrupt has occurred. If an interruption had occurred while the AIPS was on, the attention-handling routine will be effected. If no interruption had occurred, the execution of AIPS is ignored. This facility is available only to users of the assembler language; FORTRAN is self-protecting and will allow attention interruptions at any time without damage to the execution of the program. For a more detailed description of this facility, refer to Assembler Programmer's Guide.

Macro instructions in the assembler language enable the user to supply his own attention interruption-handling routines; this facility is also described in Assembler Programmer's Guide.

Conversational Task Termination

The user ends his conversational task by entering a LOGOFF command at his terminal; or he may switch his conversational execution to non-conversational (see "Switching Modes," later in this section).

Conversational Task Output

The messages produced by the system during execution of conversational tasks and the responses to command statement execution are printed at the user's terminal. The results of processing during execution of his task may be held in data sets within the system. When the user wants to examine these results, he can issue the LIST command (if the data set containing the results is a line data set) to obtain a listing at his terminal. Or he may issue one of the bulk output commands (see "Bulk Output Commands" in Part III) to print or punch the results in nonconversational mode. Also, the user can use dynamic I/O facilities in his FORTRAN and assembler language programs to obtain these results.

SYSOUT: This name designates the main task output, which includes the system messages, the responses to command execution, and the optional problem program output that are produced during execution of a user's task. In conversational mode, the information that is included in SYSOUT will be delivered at the user's terminal. The SYSOUT for a conversational task will not normally be recorded by the system in any form. Since SYSIN (see "SYSIN," above) enters and is recorded at the user's terminal, these two information streams are interspersed on the terminal listing.

Nonconversational Mode

The nonconversational mode of operation is most useful for tasks that do not require the user's presence at the terminal to resolve any problems that may arise during task execution. In this mode, there is no direct communication between the system and the user. The command statements that direct the system must have been furnished previously as a complete sequence, called a nonconversational SYSIN data set. Any system messages resulting from the execution of the task will be received by the user as printout from the central computer installation.

Nonconversational SYSIN Data Set

A nonconversational SYSIN data set is a series of command statements and associated data that are to be acted upon in the sequence in which they are presented to the system; they inform the system of the actions the user wants performed during execution of his nonconversational task. The user creates his nonconversational SYSIN data set in the same way he creates any other type of data set. He can construct it at his terminal by using the text editing commands (or DATA or MODIFY), or he can submit it on punched cards to the system operator for entry into the system via the installation's high-speed card reader. The data set must be VSAM or VISAM line, and it must be cataloged before it can be executed.

Each nonconversational SYSIN data set begins with a LOGON command and ends with a LOGOFF command, unless the mode of the task is being switched (see "Switching Modes," below). If any private input/output devices are to be used by the task, the SECURE command must immediately follow the LOGON command, preceding all requests for those devices.

Data that is to be read by the user's program during execution may be included in the SYSIN data set; this data must immediately follow the RUN command that starts execution of the user's program. For FORTRAN data sets the end-of-data record (%END) must follow the last data record.

Nonconversational Task Initiation

As in conversational mode, the user must be granted access to the system by his system administrator before attempting to communicate with the system. Then, the user can initiate his nonconversational tasks by one of these methods:

1. When the nonconversational SYSIN data set is cataloged, the user initiates his task by issuing the EXECUTE command, which must be entered from the terminal as part of his conversational task; however, EXECUTE can be given within the SYSIN data set of a nonconversational task to initiate another nonconversational task.

2. The user may start his task conversationally and then switch the mode to nonconversational, by using the BACK command (see "Switching Modes," below.)

3. By preparing his nonconversational SYSIN data set on punched cards, the user can submit it to the system operator for processing. The data set will be cataloged and execution will be requested when it is read in by the system. The user must make certain that any data sets referred to by his nonconversational task are submitted to the system before the SYSIN data set itself (see Appendix B).

Regardless of which method of nonconversational task initiation is used, the user's task is assigned a batch sequence number by the system and is executed as soon thereafter as space is available for it. The results are unpredictable if a data set affected by the nonconversational task is used before that task is finished.

The batch sequence number is a four-digit decimal number that identifies the user's nonconversational task. The user must use this number when he wants to cancel (via the CANCEL command) a previously initiated nonconversational task.

Nonconversational Task Execution

During execution of a nonconversational task, there is no communication between the system and the user. The system analyzes, in the order presented, each command of the nonconversational SYSIN data set and executes every valid command. If a command is invalid, the system will ignore it, and continue reading the SYSIN until either a valid command is read or the task is abnormally terminated. After reading and executing a valid command, the system proceeds to process the next command, continuing until it processes LOGOFF, which completes the task.

Nonconversational Task Termination

A nonconversational task is terminated in one of four ways:

1. When LOGOFF is read, normal termination occurs.

2. When the user issues the CANCEL command, specifying one of his previously initiated nonconversational tasks, that task is eliminated; a task awaiting execution can be cancelled.

3. The system terminates a nonconversational task when it encounters a situation requiring resolution by the user. Typically, such a situation arises when the system must prompt for an omitted operand in a command, or must issue a diagnostic message that requires a user response. Whenever abnormal termination of the user's task occurs, a diagnostic message that indicates the reason, will be printed as a part of SYSOUT for the task. The user may provide a special data set, identified by the DDEF command with a data definition name TSKABEND, which contains a sequence of commands to be executed if his task is abnormally terminated.

4. A system shutdown terminates all nonconversational tasks. Those initiated by PUNCH, PRINT, or WT will be automatically restarted when the system resumes operation; no restart will be attempted for other nonconversational tasks.

Nonconversational Task Output

The user specifies, by commands in his nonconversational SYSIN data set, the output expected from his nonconversational task. He must define the data sets that are to be generated and indicate how they are to be output. Other output includes printout of the SYSOUT data set, which is printed automatically by the system. This data set contains any messages issued by the system, interspersed in a listing of the commands for the task, and may also contain printable data generated by problem programs during execution of the task. All tapes, punched cards, and listings resulting from the nonconversational task are produced only at the computer center.

Note: Each SYSOUT data set begins with a message, identifying the nonconversational task and its originator.

Switching Modes

The user can use the BACK command to switch a conversational task to nonconversational task. But there is no way for him to switch from the nonconversational to conversational mode.

The user can switch his conversational task to nonconversational if all three of these conditions exist:

1. He has entered a nonconversational SYSIN data set and defined it to the system. This data set may be uncataloged; it must not begin with a LOGON command.

2. The system has space for another nonconversational task (see "Nonconversational Task Initiation," above). If not, the user will be informed and he may then try, later, to switch the mode of operation.

3. The user enters a BACK command at his terminal, requesting nonconversational continuation of his task.

If the system accepts the user's request, it will establish the nonconversational task, assign it a batch sequence number, and will eliminate the conversational task from the system. The user's terminal is then available to him for a new conversational task.

LOGON Command

This command validates the user to the system and creates the environment in which he may operate.

Operation	Operand
LOGON	user identification,[charge number],[confirmation], [message option],[password]

Note: Keyword operand format is not valid.

user identification

identifies the user to the system.

Specified as: the user identification assigned to the user at JOIN time.

charge number

specifies the user's assigned charge number.

Specified as: the charge number assigned to the user at JOIN time.
System default: the first number found in the user table for the specified user identification will be used for accounting purposes.

confirmation

this operand is included only for compatibility and has no function.

Specified as: Y or N
System default: N is assumed (even if Y is specified).

message option

this operand is included only for compatibility and has no function.

Specified as: M or C
System default: M is assumed (even if C is specified).

password

specifies the user's assigned password.

Specified as: the password assigned to the user at JOIN time.
System default: none in conversational mode if the user has been assigned a password; nonconversationally, password will not be verified.

Functional Description: The credentials the user enters (user identification, and where applicable, charge number, password) are compared with the authorization data that identify him to the system. When any or all are not valid, the user is prompted to enter all operands again. When these credentials are valid, the task continues. LOGON calls ZLOGON before control is given to the user.

Programming Notes: LOGON must precede any commands the user intends to issue. For a conversational task the LOGON command name is not entered. When the user turns on his terminal and dials the system, the system assumes a LOGON operation and immediately unlocks the keyboard for the first operand. Subsequent attempts to enter LOGON prior to LOGOFF have no effect.

If the user's permit to use the system has been withdrawn (he has been "quit") he will be advised of this via a message and his LOGON will be terminated.

Examples:

1. FRANKDOE dials up at his terminal; the system assumes that he wants to begin a conversational task.

   User: frankdoe,,,,mars7**
   System: (acknowledges that user has successfully logged on)

2. A nonconversational task is being started; the first prestored command is:

   LOGON,,,,

ZLOGON Command

This command is automatically invoked after the LOGON command is executed but before control is passed to the user. Initially ZLOGON performs no function; it allows the user to augment the initialization process.

Operation	Operand
ZLOGON	

Note: There are no operands.

Functional Description: After the task initialization process (via LOGON) is completed, ZLOGON is invoked. If the user (or the installation) has defined a procedure with the name ZLOGON, it is executed before control is passed to the user. Otherwise, ZLOGON is ignored and control is passed to the user.

Programming Notes: When the user wants some procedure to occur automatically when he logs on, he can define a procedure (via PROCDEF or BUILTIN) with the name ZLOGON or he can equate (via SYNONYM) the name ZLOGON to the name of any other command or procedure. The user can also enter ZLOGON, after it has been defined, at any time during his task.

Examples:

1. The user wants to execute program PGMA every time he initiates a task. He defines this command procedure:

```
procdef zlogon
call pgma
```

   During the LOGON process, procedure ZLOGON is invoked before the user receives control.

2. The user always wants to use the terminal card reader after initiating his conversational task. He issues:

```
synonym  zlogon=cb
```

During the LOGON process, the CB command is issued before the user gets control. CB is ignored during initiation of a nonconversational task.

TIME Command

This command establishes the time during which a task will be executed. At the end of that time, if the task is conversational, a message will be issued and control will return to the user in command mode. If the task is nonconversational, the task will be abnormally terminated.

Operation	Operand
TIME	[MINS=minutes]

MINS

specifies the number of minutes of execution time before the timer will interrupt the task.

Specified as: a decimal number greater than 0 and less than 451.
System default: the value assigned at system generation time is assumed.

Functional Description: TIME is invoked automatically as a part of the initialization of the user's task, when a time specified at system generation is used to set the timer. When the TIME command is issued by the user, the value of the timer is reset. The value of the timer will always be that of the last issued TIME command. Time is only accumulated against this interval while the user's task is actually executing. When the task is in a wait state or time-sliced out, no time is charged.

Programming Notes: The user may issue the TIME command at any time. The maximum value he can specify is 450 (7 1/2 hours).

Example: The user wants to set a four minute time limit for execution of his task.

User: time 4
System: (resets timer)

LOGOFF Command

This command notifies the system that the user wants to end his task.

Operation	Operand
LOGOFF	

Note: There are no operands.

Functional Description: LOGOFF removes the user's task from the system and releases any I/O devices used by the task. LOGOFF disposes any uncataloged data sets generated during execution of the task; disposition depends on whether the data sets reside on private storage and whether the task is in conversational or nonconversational mode.

In nonconversational mode there is no choice in the disposition. In conversational mode the user is prompted to choose the disposition. He may choose the same disposition for all his uncataloged data sets, or to have the data set names presented to him one at a time for individual disposition.

When individual disposition is selected, the user indicates his choice by entering, after the data set name is printed,

I	-	no action to be taken,
E	-	the data set is to be erased,
C [,data set name]		the data set is to be cataloged as a new data set with unlimited access (see the CATALOG command, in next section).

data set name

specifies the data set that is to be cataloged.

Specified as: a fully qualified data set name, and (optionally) a member name of a VPAM data set; when specified, the member name is enclosed in parentheses and immediately follows the VPAM data set name.
System default: the data set will be cataloged under the name just printed to identify it to the user.

When the user chooses no action for a new private data set, the system will print a list of the volumes on which the data set resides. When the user requests that a private data set be erased, the erasure will be performed only if the data set resides on a direct-access volume; otherwise, the request is ignored.

When LOGOFF is given in a nonconversational task, an automatic PRINT is issued by the system for the SYSOUT data set.

Programming Notes: LOGOFF must be the last command in every task. If no LOGOFF appears at the end of a nonconversational task, a diagnostic message is issued and the task is terminated; SYSOUT will be printed. If no LOGOFF appears at the end of a conversational task, the task will not be terminated.

Examples:

1. The user wants to end his conversational task, erase data set ARWORK2, and catalog data sets ARWORK1 and ARWORK3, all of which are uncataloged and reside on private storage.

```
User:     logoff
System:   (asks user to enter disposition option)

User:     (presses RETURN key, requesting individual disposition)
System:   ARWORK1

User:     C
System:   (catalogs ARWORK1)
          ARWORK2
```

User: e
System: (erases ARWORK2)
ARWORK3

User: c,arwork2
System: (catalogs ARWORK3 as ARWORK2 and terminates task)

2. This is the user's last command in the SYSIN data set of a nonconversational task:

   LOGOFF

   All his uncataloged private data sets are ignored, his SYSOUT is printed, and his task is terminated.

BACK Command

This command converts the user's conversational task to a nonconversational task, which takes its subsequent commands from the data set named in the command.

Operation	Operand
BACK	DSNAME=data set name

DSNAME

identifies the data set (new SYSIN) containing the series of commands that completes the current task in nonconversational mode. This data set must be cataloged or defined within the current task and currently available to the system.

Specified as: a fully qualified data set name.

Functional Description: If space for a nonconversational task is available, the user's task is accepted for execution and a batch sequence number (BSN) is assigned as soon as the BACK command is issued. Control of the task is then passed to a new SYSIN, effectively logging-off the user at the terminal. The nonconversational task takes its commands from the SYSIN data set named in the BACK operand field. The SYSIN data set should conclude with LOGOFF; if not, the system performs the LOGOFF operation and issues a diagnostic message.

If space for a nonconversational task is not available for the user's task, the BACK command is rejected, allowing the user to continue his task in conversational mode, as though he had not issued the BACK command.

A BACK command will not be accepted if the system is being shut down.

Caution: All devices needed for private volumes in a nonconversational task must be allocated via SECURE before BACK is issued.

Programming Notes: The BACK command is meaningful only in a conversational task; in a nonconversational task BACK is ignored by the system.

After issuing the BACK command, the user must go through the normal task initiation process to begin a new conversational task.

When BACK is rejected, the user may reissue it at some later time during his terminal session, but he must be aware of the necessity for modifying the new SYSIN data set to reflect his conversational processing subsequent to his first BACK command.

If the user, when he issues the BACK command, interrupts a program that is being executed, the first command in his SYSIN data set should be GO, which will cause program execution to resume at the point of interruption.

When the user wants to initiate a nonconversational task that does not require a prior conversational phase, he should use the EXECUTE command. The data set named as SYSIN in the EXECUTE command, unlike that named in the BACK command, must begin with LOGON and conclude with LOGOFF, and must be on public storage.

Example: The user wants to change his conversational task to nonconversational, using data set ALPHA as SYSIN for his nonconversational task.

User: back alpha
System: (accepts task and assigns batch sequence number)

EXECUTE Command

This command introduces a nonconversational task into the system.

Operation	Operand
EXECUTE	DSNAME=data set name

DSNAME

identifies the data set which resides on public storage and which contains a series of commands that start with LOGON and end with LOGOFF; this data set becomes the SYSIN of the nonconversational task.

Specified as: a fully qualified data set name.

Functional Description: EXECUTE requests creation of a nonconversational task that is independent of the user's current tasks. A batch sequence number is assigned and the task is created when task space becomes available.

Programming Notes: The nonconversational task is controlled by the commands in the SYSIN data set, starting with LOGON and ending with LOGOFF. Each SYSIN data set represents one task only.

The EXECUTE command differs from the BACK command:

1. EXECUTE requests an independent nonconversational task, rather than changing the user's conversational task to nonconversational mode.

2. The data set named in the EXECUTE command must contain LOGON and LOGOFF commands; the data set specified in the BACK command need only conclude with a LOGOFF command.

3. EXECUTE is accepted by the system even if no task space is currently available; the task will be created later. If task space is not available when the BACK command is issued, the command is ignored and the user continues conversational processing as though he had not issued the command.

Example: The user wants to create a nonconversational task; the commands for the task are stored as a data set named NEWTASK.

User: execute newtask
System: (accepts task and assigns a batch sequence number)

SECURE Command

This command reserves all devices that will be required for private volumes during the execution of a nonconversational task.

Operation	Operand
SECURE	{ (TA = number of devices [,type of device]) / (DA = number of devices [,type of device]) } [,...]

Note: TA and DA must be specified in keyword format.

TA

designates the number of tape devices requested.

Specified as: a one- or two-digit number.

identifies the type of tape device requested.

Specified as:

7 -- seven-track tape

7DC -- seven-track tape with data converter feature

9 -- nine-track tape

System default: the type of tape specified at system generation time is assumed.

System default: no tape devices are reserved.

DA

designates the number of direct-access devices requested.

Specified as: a one- or two-digit number.

identifies the type of direct-access device.

Specified as:

2311 -- disk

2314 -- disk

System default: the type of direct-access device specified at system generation time is assumed.

System default: no direct-access devices are reserved.

Note: One of the operands (TA or DA) must be specified.

Functional Description: SECURE reserves the specified devices as a group so that the task can proceed, without pause, when the entire group is available. Any waiting for devices occurs when the SECURE command is being executed.

Cautions: If SECURE is not given where required, or an error exists in the command itself, the user's task is terminated.

The user must provide a SECURE command immediately after the LOGON command in every data set that is to be executed as a separate nonconversational task and that refers to one or more private volumes.

Programming Notes: SECURE applies to nonconversational tasks only; it is never executed in a conversational task.

Examples:

1. The user has prepared a task for nonconversational execution that includes references to private volumes. For these, one 2311 disk drive and three tape units, all for 9-track tape, are necessary. He prepares this SECURE command for insertion immediately after the LOGON command:

```
secure  (ta=3,9),(da=1,2311)
```

2. The user's nonconversational task requires three 2311 disk drives and seven tape units, three of them 7-track and the remainder 9-track. He prepares this SECURE command:

```
secure (da=3,2311),(ta=3,7),(ta=4,9)
```

CANCEL Command

This command eliminates a nonconversational task or job that is either waiting for task space or is currently being executed.

Operation	Operand
CANCEL	BSN=batch sequence number

BSN

identifies the nonconversational task to be canceled.

Specified as: the one- to four-digit batch sequence number assigned by the system when the nonconversational task was established.

Functional Description: When a task is canceled during its execution, the devices reserved for its use are released, and the pages of storage it was using are freed; the SYSOUT, although probably incomplete, will be printed and will include a message indicating the reason for task termination.

A task canceled before it starts execution receives no explicit sign of cancellation.

The user will be informed if the task designated in the operand field cannot be found.

Programming Notes: The user may cancel any of his nonconversational tasks including those initiated through the bulk output commands.

Example: The user wants to cancel a nonconversational task (before execution) with a batch sequence number of 1214.

```
User:     cancel bsn = 1214
System:   (cancels nonconversational task)
```

ABEND Command

This command returns the user's task to a status that existed after the LOGON process.

Operation	Operand
ABEND	

Note: There are no operands.

Functional Description: When ABEND is executed, the user's current task is terminated and a new task is created, as if the user had logged on again. All data set definitions, open data sets, and session variables are eliminated.

Programming Notes: The ABEND command can either be entered from the terminal or automatically induced by the user pressing the ATTENTION key on his terminal five consecutive times with no intervening activity on his part.

This command is intended for use in situations where other recovery procedures are not adequate, and the user must get out of an unresolvable situation.

Example: The user, while executing a program, is in an unrecoverable situation:

```
User:     (presses ATTENTION key once)
System:   !

User:     abend
System:   ("cleans up" virtual storage)
```

The facilities available through commands, which provide the user with convenient means to manage his data sets, are:

- Data set management
- Text editing
- Data editing
- Bulk output

The data set management commands provide for identifying data sets; for efficiently storing them within the system and retrieving them; for sharing them with other users; for copying and erasing them; and for defining them and their use in the system. These commands are described in Section 1.

The text editing commands enable the user to create and edit VISAM line and VISAM region data sets. The text editor and its commands are described in Section 2.

The data editing commands are used to build and edit VSAM and VISAM data sets, although these facilities are less flexible than those of the text editor. These commands are described in Section 3.

The bulk output commands allow the user to establish independent non-conversational tasks, to efficiently output his data sets. These commands are described in Section 4.

SECTION 1: DATA SET MANAGEMENT

The data set management commands and the system functions they request are shown in Table 4.

Table 4. Data Set Management Commands and Their Functions (The commands are listed in the same order as the command descriptions that follow.)

Command	Function
CATALOG	Create or alter catalog entry for data set; create catalog index for generation data group; or catalog data set as new generation of existing generation data group.
DDEF	Define data set and describe its characteristics to system.
RELEASE	Delete data definition established by previous DDEF command.
CDD	Retrieve DDEF commands, which have been prestored in cataloged or defined line data set, and process them.
DELETE	Delete one data set entry from user's catalog.
ERASE	Free direct-access storage assigned to private data set and remove its catalog entry from user's catalog.
PERMIT	Authorize or withdraw authorization of other users to access user's specified data set.
SHARE	Allow user to share data sets belonging to another user who has granted authorization with PERMIT command.
DSS?	Present status of cataloged data sets.
POD?	Display information about members of VPAM data set.
VT	Copy VAM data set onto tape as physical sequential data set.
TV	Retrieve data set that was written onto tape via VT command and write data set into VAM volume.
VV	Copy VAM data set on direct-access storage.
CDS	Duplicate data set or member of VPAM data set.

CATALOG Command

This command creates or alters the catalog entry for a data set, creates a catalog index for a generation data group, or catalogs a data set as a new generation of an existing generation data group.

The CATALOG command, depending on the objective, takes one of two forms:

Form 1

Operation	Operand
CATALOG	DSNAME=data set name [,STATE={N|U}] [,ACC={R|U}] [,NEWNAME=data set name]

Form 2

Operation	Operand
CATALOG	GDG=data set name,GNO=number of generations [,ACTION={A|O}] [,DISP={E|S}]

DSNAME

identifies the data set which must be already defined by a DDEF command within the current task or must be cataloged, and which resides on a direct-access or magnetic-tape volume.

Specified as: a fully qualified data set name, which must not have an absolute generation number appended.

STATE

specifies whether this is the updating of an existing catalog entry or the creation of a new catalog entry.

Specified as: N - new
U - update
System default: N is assumed.

ACC

specifies the access qualification for the data set.

Specified as: R - read-only
U - unlimited
System default: U is assumed if the catalog entry is new; otherwise, no change will be made to the access qualification.

NEWNAME

designates the new name for the data set.

Specified as: a fully qualified data set name.
System default: the data set name is unchanged.

GDG

identifies a new generation data group.

Specified as: a generation data group name; maximum number of characters, 26.

Note: This operand must be given in keyword format.

GNO

indicates the number of generations to be maintained in the generation data group.

Specified as: a one- to three-digit decimal number; maximum value, 255.

ACTION

specifies the action to be taken when the GNO value plus one generation is being cataloged in the generation data group.

Specified as: A - all previous generations to be removed from catalog.
O - only oldest generation to be removed.
System default: O is assumed.

DISP

designates the disposition of old generations deleted from the catalog. Disposition applies to private volumes only; public data sets are always erased when uncataloged.

Specified as: E - old generation data sets to be erased from storage.
S - old generation data sets to be saved.
System default: S is assumed.

Functional Description: CATALOG offers two options: (1) to create a catalog entry for a data set (Form 1) or a generation data group (Form 2), or (2) to update the catalog entry for a data set (Form 1).

When the user wants to catalog a data set, the system enters the specified data set name into the user's catalog, and assigns to the data set the access qualification specified by the user. If a data set name is specified with a member name, the data set name, not the appended member name, is cataloged.

When the user wants to catalog a generation data group, the system enters the generation data group name into the user's catalog, and stores information pertaining to the maximum number of generations to be maintained, what is to happen when that number is exceeded, and the disposition of deleted generations.

The user can catalog a generation to the generation data group with either an absolute or relative generation number. When the relative number is used, the system automatically assigns the proper absolute generation number to the generation and prints that number. When the user catalogs a generation to a generation data group and exceeds the maximum number of generations maintained, the system removes all, or the oldest generation, depending on the option selected by the user.

To update a catalog entry, the user specifies as DSNAME, the name by which the data set is currently defined (or cataloged). If NEWNAME is specified, the system changes the data set labels (DSCBs) of the direct-access volumes containing the data set, thus carrying through the name change.

Caution: The user should not rename data sets that reside on magnetic-tape volumes; the user may lose data sets that reside on magnetic tape if he renames them in his catalog.

Programming Notes: The access qualification that the user specifies in creating or updating a catalog entry for a data set applies only to his own access to his data set; it has no effect on the sharing or permitting of that data set. Furthermore, read-only qualification does not hinder the user from erasing his own data sets whenever he chooses.

All VSAM, VISAM, and VPAM data sets in public storage are immediately cataloged when a DDEF command is initially issued for the data set. At initial DDEF time, the system creates a catalog entry and provides the user with read/write access.

Once a generation data group has been cataloged, the user can add generations to the group. When he builds the data set, the initial DDEF command will automatically catalog the data set as a new generation of the specified generation data group.

A CATALOG command cannot be used to update the entry for a generation data group. CATALOG can, however, update a generation of a generation data group. Example: To change the version portion of an absolute generation number, the user may issue a CATALOG command, giving the current name of the generation (including its relative or absolute generation number) as DSNAME. He then specifies the new generation name, with its updated absolute generation number, as NEWNAME.

The user can catalog a new generation of a generation data group with either an absolute or relative generation number; any cataloged generation can be referred to by either generation number. When using relative numbers, the user must be responsible for knowing the actual generation being referenced. The newest generation is relative generation number 0.

Only the owner of a generation data group is allowed to catalog generations of that group. Sharers, regardless of their level of access, are not permitted to do this.

A user who has been granted unlimited sharing access to another's catalog may add entries to that catalog. When naming such entries, the user must include qualifiers with the same names that he assigned in his SHARE command for that catalog. Similarly, if he wants to rename such an entry, he must include the SHARE qualifier as a part of the new name. If the user does not give the SHARE qualifier, the system cannot tell which catalog is desired and issues a diagnostic message.

When cataloging a new data set, the user can specify a second name (NEWNAME) and that name is assigned to the catalog entry. For example, a data set created under System/360 Operating System may bear an overlength name; by means of NEWNAME, the user can rename it to suit TSS/360 requirements.

Examples:

1. The user wants to recatalog direct-access data set X.X2 under the new name SMK.SIMUL with read-only access:

   User: catalog x.x2,u,r,smk.simul
   System: (changes catalog entry and DSCB name to SMK.SIMUL)

2. The user wants to catalog ASET as a new 10-generation data group. By defaults, he also wants the new group operands to indicate that only the oldest generation is to be removed and saved when the eleventh (i.e., GNO+1) generation is cataloged:

   User: catalog gno=10,gdg=aset
   System: (creates catalog entry)

3. The user now wants to catalog a new generation of generation data group ASET. It is assumed that the generation has been previously defined:

   User: catalog xgz,u,u,aset(+1)
   System: (creates catalog entry for generation)

   Note: The system automatically issues the absolute generation number assigned to the generation. The user may then refer to that generation by absolute generation number or by relative generation number. The relative generation number of the most recently cataloged generation is always 0.

4. In a subsequent task, the user wants to catalog another new generation of generation data group ASET. The new generation is assumed to have been defined:

   User: catalog abc,u,u,aset(+1)
   System: (creates catalog entry for generation)

   Note: The relative generation number correlates with the next-available absolute generation number. The user must know the relationship between relative and absolute generation numbers whenever he uses relative generation numbers. However, he can always refer to generations by relative generation numbers.

5. The user wants to catalog another new generation of generation data group ASET. This new generation has been created and defined as LATE3:

   User: catalog late3,n,u,aset.g0003v00
   System: (creates catalog entry for generation)

   Note: The user now must issue a DDEF command to define the new generation before he can refer to it. This DDEF command defines either ASET.G0003V00 or ASET(0).

6. A user (User 2) wants to add the previously defined private data set DO.FILE.B4 from the owner's (User1) catalog. User1 issues a PERMIT command to grant User2 unlimited access to the entire catalog. User2 in a SHARE command, assigns the name DO to this catalog; he catalogs the new data set with unlimited access.

   User1: permit *all,n,u,user2
   User2: share do,user1,ownerds=*all
          ddef ddni,dsname=do.fire.b4,disp=new
          catalog do.fire.b4,n,u
   System: (creates catalog entry for DO.FIRE.B4 in User2's catalog)

DDEF (Data Set Definition) Command

This command defines a data set and describes its characteristics to the system; and catalogs public VSAM, VISAM, and VPAM data sets. In general, any data set that will be referenced by an object program during execution must be defined by a DDEF command.

Note: The operand list for the DDEF command is extensive; the complete list is in Appendix F. The DDEF command is shown below in its expected normal form for typical public VAM data sets; in this normal usage it is expected that most or all of the other operands will be defaulted.

Operation	Operand
DDEF	DDNAME=data definition name,[DSORG={VI|VS|VP}], DSNAME=data set name

DDNAME

specifies the symbolic data definiton name that is associated with the data set, and which provides a link between the data control block (DCB) in the user's program and the data set definition.

Specified as: one-to-eight alphameric characters, the first of which must be alphabetic.

DSORG

indicates the organization of the data set being defined.

Specified as: VI - VISAM
VS - VSAM
VP - VPAM
System default: VI is assumed.

DSNAME

specifies the name by which the data set will be cataloged and referred to during the current task.

Specified as: a fully qualified data set name and (optionally) a member name of VPAM data set; when specified, the member name is enclosed in parentheses and immediately follows the VPAM data set name.

Functional Description: The DDEF command establishes a system entry for the data set definition that can be referenced by allocation routines and access methods. This link between the data set definition and the problem program's reference to the data set (i.e., the data control block) is the data definition name. The entry containing the data set definition is maintained until the task is concluded or the definition is deleted via the RELEASE command.

All VSAM, VISAM, and VPAM data sets in public storage are immediately cataloged when initially referenced by a DDEF command. At initial DDEF time, the system creates the catalog entry and provides the user with read/write access.

When a member name is specified as DSNAME, DDEF defines the VPAM data set, not the member.

Caution: If a user's program is executing in conversational mode and refers to an undefined data definition name, a diagnostic message will be issued; in nonconversational mode, the task will be abnormally terminated. A FORTRAN user can default his terminal as the "undefined" data source or destination.

Programming Notes: Each DDEF command is valid only during the session in which it was issued; previously defined data sets must be redefined in each new session that references them.

The user can change the data definition name assigned in a previous DDEF command by issuing DDEF with the new data definition name; the only operands required are data definition name and data set name. The new data definition name will then be assigned; the old one will be eliminated.

Examples:

1. The user wants to define a VPAM data set.

   User: ddef ddn1,vp,dsname=group(mem1)
   System: (establishes entry for data set definition)

   This command defines the VPAM data set GROUP, not the member (MEM1). A DDEF for GROUP (MEM2) will not create a new data set definition, but will only replace the data definition name in the previous DDEF for GROUP (MEM1).

2. The user wants to define a new VISAM public data set.

   User: ddef ddn2,,t.trip3
   System: (establishes entry for data set definition)

RELEASE Command

This command deletes the data set definition established by a previously issued DDEF command. RELEASE will cancel a preceding DDEF command for either a public or private data set; also, it will release the I/O devices associated with a private data set. It may also be used to deconcatenate and release one or all data sets of a given concatenation (see detailed description of DDEF in Appendix F for concatenation), or to remove JOBLIBs from the user's program library list.

Operation	Operand
RELEASE	DDNAME=data definition name[,DSNAME=data set name]

DDNAME

identifies the data set definition created by a DDEF command that was issued earlier in the current task. The name either identifies the data set definition to be released or identifies the concatenation from which one or all data sets are to be deconcatenated and released.

Specified as: the data definition name specified in a previous DDEF command.

DSNAME

identifies one data set in a concatenated series. Only this data set is to be released; the remainder of the concatenation is not affected.

Specified as: a fully qualified data set name.
System default: all data sets concatenated with the specified data definition name are released.

Note: This operand is used only for concatenated data sets, and has no meaning in any other situation.

Functional Description: RELEASE deletes the information defining the data set and frees for other use all I/O devices currently assigned to the specified private data set. If the data set is open, it will be closed before the defining information is deleted.

When the specified data definition name applies to the data set definition of a JOBLIB, the JOBLIB is removed from the program library list, the definition of the data set is deleted, and any associated I/O devices are released.

When the data set name of a concatenated data set is specified, that data set is released and dropped from the concatenation. The rest of the concatenation remains unchanged and may still be referenced by its data definition name. If DDNAME refers to a concatenation and DSNAME is not specified, all data sets in the concatenation are released.

When there is more than one data set on the private volume being released, the device that contains the volume is not released until a RELEASE command has been issued for the last data set on that volume.

When the user specifies DDNAME for a data set on a public volume, the definition of the data set is deleted, but the device is not released.

Programming Notes: The RELEASE command does not erase or uncatalog; it deletes the current definition of a data set and frees any I/O devices assigned to the private volume on which that data set resides. The user, therefore, should issue a RELEASE command when a data set is no longer needed in a task. He must redefine the released data set when he wishes to refer to it again.

Examples:

1. The user wants to release a private data set identified by DDNAME INGO.

   User: release ingo
   System: (deletes the current definition of the data set and frees the I/O devices assigned to the private volume)

2. The user wants to release a concatenated data set (with DDNAME TABLES) that has three data sets (DTAB1, DTAB2, and DTAB3).

   User: release tables
   System: (releases the concatenation)

3. The user wants to release a concatenated data set (TURN9) from a concatenation with DDNAME OVERT.

   User: release overt, turn9
   System: (releases TURN9 from the concatenation)

4. The user wants to release a job library, PROGTEST.

   User: release progtest
   System: (removes the JOBLIB from the program library list)

CDD (Call Data Set Definition) Command

This command retrieves one or more DDEF commands that have been prestored in a cataloged or defined line data set and then processes those commands.

Operation	Operand
CDD	DSNAME=data set name [,{data definition name / (data definition name,...)}]

DSNAME

identifies the line data set, which contains prestored DDEF commands, and which must be defined by a DDEF command within the current task or must be cataloged.

Specified as: a fully qualified data set name.

data definition name

identifies the particular DDEF commands to be retrieved in the referenced data set.

Specified as: the data definition name or names of the DDEF commands to be retrieved. When two or more data definition names are entered, they must be enclosed in parentheses.

System default: all DDEF commands in the referenced data set are to be retrieved.

Note: This operand must be specified by position.

Functional Description: the CDD command retrieves one or more DDEF commands from the specified data set and then processes them. The user can thus create a cataloged line data set of commonly used DDEF commands and refer to them by the CDD command, thereby relieving himself of direct DDEF command entry. Each DDEF command that is executed is printed out in full. Any incorrect DDEF commands are also printed.

Cautions: Each data definition name must be unique within the task. The prestored data set must contain DDEF commands only. A diagnostic message is issued if data or any other command appears in the data set. The conversational user has the option of either skipping the erroneous records in the data set or cancelling the CDD command; a nonconversational task is terminated.

The user must verify that the prestored DDEF commands are correct; when these commands are executed via the CDD command, the messages normally issued to the user for DDEF will not be issued.

Programming Notes: The user can retrieve and enter all prestored DDEF commands in the data set by omitting the data definition name operand. If the user wants to retrieve a selected set of these commands, he must supply the data definition names of the selected DDEF commands when he enters the CDD command.

Examples:

1. The user wants to execute three DDEF commands stored in cataloged line data set PAYROLL.DD. The three DDEF commands with data definition names NOW1, NOW2, and NOW3, are assumed to be in the data set.

```
User:     cdd payroll.dd,(now1,now2,now3)
System:   (processes DDEF commands in data set; then
          prints information similar to)
          DDEF NOW1,VI,DSNAME=WINDUP,DISP=OLD
          DDEF NOW2,VI,DSNAME=GOONNU,DISP=OLD
          DDEF NOW3,VS,DSNAME=STAR,DISP=OLD
```

2. The user wants to execute the DDEF command with DDNAME JBACCT in data set PAYROLL.P.

   User: cdd payroll.p,jbacct
   System: (processes specified DDEF command, then prints information similar to)
   DDEF JBACCT,PS,DSNAME=LEAD.T,DCB=(DEN=2),
   UNIT=(TA,9),VOLUME=(,043591),LABEL=(2,SL,RETPD=2),
   DISP=OLD

DELETE Command

This command deletes a data set entry from the user's catalog. DELETE is intended primarily for data sets on private volumes, but the user may use it to remove from his catalog entries for any public data sets that he is sharing.

Operation	Operand
DELETE	DSNAME=data set name

DSNAME

identifies the data set which resides on a private volume, or the shared data set owned by another user, which is to be deleted.

Specified as: a fully qualified data set name, a name of a generation data group, or a partially qualified data set name.

Functional Description: When the data set name specified is partially qualified, all data sets indexed under that name are found in the catalog. In conversational mode, these names are listed at the terminal, and the names of private data sets are deleted; the names of public data sets are not deleted, and a diagnostic message is issued. In nonconversational mode, the catalog entries of all specified data sets on private volumes are deleted.

Programming Notes: When the user wants to delete the catalog entry for a public data set and free the space a data set occupies, he must use the ERASE command.

To delete a generation data group, each of its members must be recataloged as a nonmember of the group prior to execution of the DELETE command.

Examples:

1. The user wants to delete data set R.RECORD.

   User: delete r.record
   System: (informs the user that the catalog entry for R.RECORD has been deleted)

2. The user wants to delete three data sets (A.B.Y, A.B.Z, and A.B.X), using the partially qualified data set name A.B:

   User: delete a.b
   System: (informs the user that the catalog entries for A.B.Y, A.B.Z, and A.B.X have been deleted)

3. The user attempts to delete his data set WEARLON, which resides on a public volume.

   User: delete wearlon
   System: (responds by informing the user that WEARLON resides on public storage and cannot be deleted, and suggests the user use the ERASE command)

ERASE Command

This command frees the direct-access storage assigned to a data set. Also the entry for a cataloged data set is removed from the user's catalog.

Operation	Operand
ERASE	DSNAME=data set name

DSNAME

identifies the data set to be erased; this data set, which must reside on direct-access storage, must already be defined by a DDEF command within the current task, or must be cataloged.

Specified as: a partially qualified data set name, or a fully qualified data set name and (optionally) a member name of a VPAM data set; when specified, the member name is enclosed in parentheses and immediately follows the VPAM data set name.

Functional Description: When the DSNAME operand is not a member name, the direct-access storage occupied by that data set is freed (i.e., released for other use), and the data set name is removed from the catalog.

When the DSNAME operand specifies a member name of a VPAM data set, the member name is deleted from the partitioned organization directory (POD) and the storage occupied by the member is freed.

When the DSNAME operand specifies a partially qualified data set name, all data sets indexed under that name are found in the catalog. Storage space for these data sets is freed and their catalog entries are removed. When a VPAM data set name is supplied without a member name, the storage for the entire data set is freed and its name is removed from the catalog.

In conversational mode, the names of erased data sets are displayed at the terminal.

Any previous DDEF command, issued on a data set that is erased, is also erased.

When a user attempts to erase a shared VISAM or VPAM data set, the system checks for any active users (including the one who issued ERASE) of that data set.

- If there are active users, the system issues a diagnostic message and disregards the ERASE command. In conversational mode, the diagnostic message appears at the terminal, followed by an underscore requesting the next command. In nonconversational mode, the new command is retrieved from the task's SYSIN after the diagnostic message is sent to SYSOUT.
- If there are no active users, the ERASE command is executed.

Cautions: The ERASE command cannot be used for data sets on magnetic tape; it applies only to data sets on direct-access storage.

The user should not issue an ERASE command for a loaded module; the module should be unloaded first.

In nonconversational mode, the SYSIN data set cannot contain an ERASE of itself.

Examples:

1. The user wants to free the storage occupied by data sets whose names begin with the components A.B.

   User: erase a.b
   System: (frees the storage space, removes the catalog entries, and displays a list of the names of the erased data sets)

2. The user wants to free the storage occupied by data set GN.PYKK1 GN.PHLL1.

   User: erase gn.pykk1
   System: (frees the storage allotted to GN.PYKK1, removes its entry from the catalog, and displays the name of the erased data set)

3. The nonconversational user wants to free the storage occupied by member EXPON002 of VPAM data set LIBPROG.

   User: erase libprog(expon002)
   System: (frees the storage occupied by the member and erases its name from the POD)

PERMIT Command

This command allows the user to permit or restrict sharing of his cataloged data sets by other users.

Operation	Operand
PERMIT	DSNAME={data set name | *ALL}[,STATE={N | U}], [,ACCESS=access qualification] [,USERID={(user identification[,...]) | *ALL}]

DSNAME

identifies the cataloged data set for which sharing is being permitted or restricted.

Specified as: a partially or fully qualified data set name.
*ALL - all cataloged data sets of the user are to be shared. (This is referred to as sharing of the catalog.)

STATE

specifies whether this is a new list of sharers or an update of a previously issued list.

Specified as: N - new list (i.e., no previous PERMIT for this data set)
U - update of existing list
System default: N is assumed.

ACCESS

designates the access qualification for users sharing the data sets.

Specified as:

R - restricts (i.e., withdraws) sharing access that was previously permitted.
RO - read-only access; sharers may only read the data set.
RW - read-and-write access; sharers may both read from and write to the specified data set but may not erase it.
U - unlimited access; sharers may read, write, and erase the data set.

System default: when STATE = N is specified, U is assumed; when STATE = U is specified, the existing qualifier as supplied in the previous PERMIT is assumed.

USERID

identifies the user being permitted or restricted sharing of the specified data set.

Specified as: the user identification of one or more permitted or restricted users.
*ALL -- all users of the system are permitted or restricted sharing.

System default: *ALL is assumed.

Functional Description: When PERMIT is issued to permit sharing, the system either (1) enters the list of sharing - user identifications, and the associated access qualifiers, in the owner's catalog entry that was specified by the DSNAME operand, or (2) marks that catalog entry for universal sharing. These notations are made only in the owner's catalog; sharers' catalogs are unaffected by the PERMIT command.

When PERMIT is issued to restrict sharing, entries for sharers are removed from the owner's catalog entry.

If a PERMIT command is repeated, with identical information, the repetition is ignored.

Cautions: If a sharer erases a data set to which he has been given unlimited access, the entry for that data set is also removed from the owner's catalog. Thus, the owner's catalog can be changed without his knowledge.

The owner of a shared data set cannot withdraw sharing privilege from an active user of that data set.

After a PERMIT command is issued for a data set (or a job library), the original data set definition is not changed (i.e., it indicates private ownership). If the permitted data set is to be processed in the same session in which PERMIT was issued, the owner should release the existing data set definition and issue a new one. If the name of the original data set definition is known, he can use the RELEASE command to release the data set definition; otherwise, he can use LOGOFF. The new data set definition is created with DDEF.

Programming Notes: After a PERMIT command is issued, the designated sharers must issue SHARE commands to link their catalog entries to the owner's. Only then can sharers reference the data set under the owner's catalog entry.

Once the owner grants access to all other users, he must also restrict all users before he can selectively change the access qualification for a specific user. Example: If all users have previously been granted access to catalog entry MB.C and the owner now wants to restrict every user, except SSIMON and LAF29, he must first restrict all users:

```
permit mb.c,u,r,*all
```

This marks catalog entry MB.C as private. Since the entry is now private, the PERMIT command to grant SSIMON and LAF29 access creates a new list of sharers and, therefore, must specify the N state:

```
permit mb.c,n,rw,(ssimon,laf29)
```

Similarly, to grant a sharer access to a lower-level catalog entry than he was previously permitted, the owner must restrict the sharer from the level now permitted him. For instance, user JOE7 was allowed access to catalog entry BOOK.A and the owner now wants to change this level so that JOE7 can refer only to the lower-level catalog entry BOOK.A.F.X. Then the first PERMIT command must be issued to restrict JOE7 from his present level:

```
permit book.a,u,r,(joe7)
```

This removes JOE7 from the sharer list for catalog entry BOOK.A. To allow JOE7 to share the lower-level catalog entry, the owner now issues:

```
permit book.a.f.x,n,ro,(joe7)
```

The access qualification granted to a sharer is not limited by the access level established for the owner during cataloging. For instance, the owner can catalog a data set with read-only access for himself and still assign unlimited access to a sharer in a PERMIT command.

Examples:

1. The user wants to allow users JOSEPH24 and HENRY24A to share his cataloged data set AD.AT1 with read-only access. These are the only sharers in the sharer list.

   ```
   User:     permit ad.at1,n,ro,(joseph24,henry24a)
   System:   (enters list of sharers in owner's catalog)
   ```

2. The user now wants to update the list created in Example 1 by changing the access of users JOSEPH24 and HENRY24A to read/write.

   ```
   User:     permit ad.at1,u,rw(joseph24,henry24a)
   System:   (updates sharing list)
   ```

3. A user wants to restrict all users from all of his cataloged data sets.

   ```
   User:     permit *all,n,r
   System:   (removes sharing list from owner's catalog)
   ```

   Note: Each data set is automatically restricted from all other users until the owner grants access, through a PERMIT command, to one or more sharers.

4. A user wants to share his object modules in his user library with JBROWN#1.

   ```
   User:     permit userlib,n,ro,(JBROWN#1)
   System:   (enters sharing list in owner's catalog)
   ```

SHARE Command

This command allows the user to share another user's (the owner) data sets. The data set owner must previously have granted the user permission, by a PERMIT command, to share the data set.

Operation	Operand
SHARE	DSNAME=data set name, USERID = owner's user identification [,OWNERDS= {owner's data set name | *ALL}]

DSNAME

specifies the name by which the sharing user refers to the data set or data sets to which he has been granted access by a PERMIT command. This data set name becomes an entry in the sharer's catalog.

Specified as: a fully or partially qualified data set name.

USERID

identifies the data set owner (i.e., the user who issued the PERMIT command).

Specified as: the owner's user identification.

OWNERDS

identifies the data sets to which the user wants access.

Specified as: the fully or partially qualified data set name assigned by the owner;
*All - the user wants access to all the owner's cataloged data sets.
System default: *ALL is assumed.

Functional Description: The owner's catalog is searched to determine whether the entry given in the OWNERDS operand can be shared by the user who initiated the SHARE command. If so, an entry is made in the sharer's catalog, under the data set name specified by DSNAME, pointing to the owner's catalog entry for OWNERDS. The pointer is to the initial entry in the owner's catalog if "*ALL" is specified.

If the user cannot share the owner's catalog entry (i.e., the owner did not issue a PERMIT command for him), the SHARE command is ignored and a diagnostic message is issued.

Cautions: The OWNERDS operand must have the same value as the DSNAME operand the owner used when issuing his PERMIT command.

To avoid the possibility of violating the length restriction for data set names, the sharer should not enter a DSNAME operand that is longer than the OWNERDS operand. Similarly, if "*ALL" is used, the sharer must be certain that the total number of characters for DSNAME plus any data set name in the owner's catalog does not exceed 35.

Programming Notes: When OWNERDS in the owner's catalog is a partially qualified data set name, the sharer refers to each shared data set by appending to the data set name, specified by DSNAME, the same rightmost name or names that the owner assigned (in his catalog) to that data set. Example: If OWNERDS specifies a catalog entry for the partially qualified data set name A.B, and the sharer gives W.X in the DSNAME operand, he refers to the owner's data set A.B.C.D as W.X.C.D.

The sharer's catalog entry for a shared data set is not removed when the owner erases or deletes that data set from his own catalog. Sharers must update their own catalogs by using the DELETE command.

Examples:

1. The user wants to reference, by means of the name GREYX, the catalog entry for data set M.LOG1, to which he has been granted access by owner MICHAEL2.

   User: share greyx, michael2, m.log1
   System: (makes entry in sharer's catalog)

2. The user has been granted access to all of owner JOSEPH24's cataloged data sets. He wants to use the name Z to link his catalog to the initial entry in JOSEPH24's catalog.

   User: share z,joseph24, *all
   System: (makes entry in sharer's catalog)

   The user now can reference specific data sets belonging to JOSEPH24. For instance, if JOSEPH24's catalog has data sets named A.A, A.B, and A.C, the user refers to them as Z.A.A, Z.A.B, and Z.A.C, respectively.

DSS? (Data Set Status) Command

This command presents the status of one or more cataloged data sets to the user.

Operation	Operand
DSS?	[NAMES = { data set name / (data set name,...) }]

NAMES

identifies one or more cataloged data sets for which status information is to be presented.

Specified as: one or more fully or partially qualified data set names.
System default: the status of every data set in user's catalog is presented.

Note: When this operand specifies a VPAM data set, only the status of the VPAM data set is given, not that of each member.

Functional Description: DSS? provides the user with this information about a data set:

Sharing status - ownership and shareability

Access status - read-only, read-write, or unlimited

Device type and volume number

Creation and expiration dates

Organization

For VAM data sets only, the date last used and the length of the data set

If a partially qualified data set name is specified, the status of each data set with the specified qualifiers is presented.

Sharing status is given only for those data sets that are permitted (via the PERMIT command) under their fully qualified names; no sharing status is given if a partially qualified data set name or the user's entire catalog is permitted. In nonconversational tasks, the status information is recorded on SYSOUT; in conversational tasks, the information is printed on the user's keyboard, but the user can terminate the printing at any time by pressing the ATTENTION key.

Examples:

1. The user wants to present the status of data set DATAR.

   User: dss? names=datar
   System: (presents status information)

2. The user wants to present the status of all data sets qualified by D.A.

   User: dss? d.a.
   System: (presents status information)

POD? Command

This command places on SYSOUT a list of the member names and (optionally) the alias names and other information pertaining to individual members of cataloged VPAM data sets.

Operation	Operand
POD?	DSNAME=data set name [,H] [,A]

Note: H and A are not positional operands; they may be specified in either order.

DSNAME

identifies the cataloged VPAM data set for which member information is to be presented.

Specified as: the fully qualified name of a VPAM data set, or the absolute or relative generation name of a VPAM member of a generation data group.

H

specifies that any system and user data associated with each member is to be printed in hexadecimal. Only the first 21 bytes (42 hexadecimal digits) of user data are printed. The format and content of this data is user-defined. Similarly only certain system data (25 hexadecimal digits) can be printed.

Specified as: H
System default: the system and the user data associated with each member is not printed.

A

specifies that any aliases of each member are to be printed. An alias is another name by which a member of a VPAM data set can be identified.

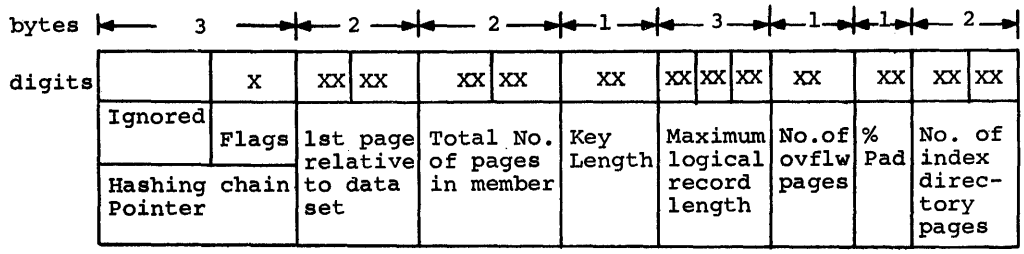
Specified as: A
System default: the members' aliases are not listed.

Functional Description: If a VPAM data set is a program library (e.g., USERLIB or a job library), its members are object program modules. Each member has a name that was assigned by the user during compilation, assembly, or link editing. This name is used by the system as the basis for stowing the module in the library, and it is recorded in the program library's directory (POD). To make each module available on the basis of other names (e.g., entry point names), the system also defines a number of aliases for the module (e.g., all external symbol definitions except named COMMON are defined as aliases). The alias names are also stored in the POD by the system. The user can thus invoke a module based on its member name or any of its aliases.

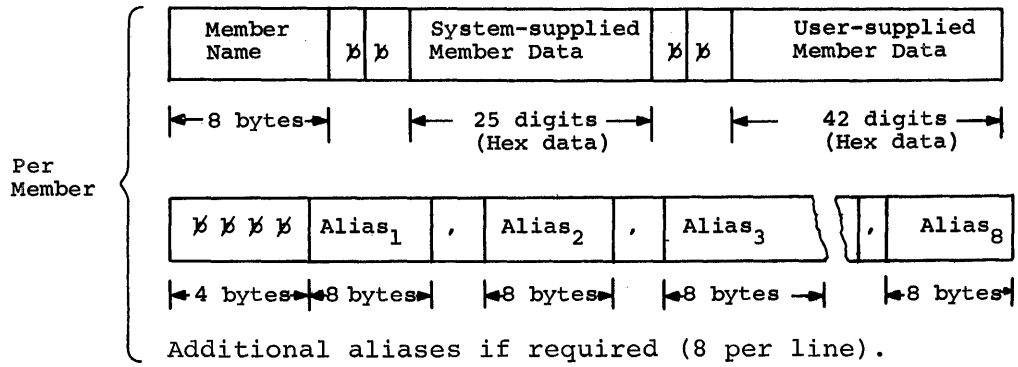
If a VPAM data set is not a program library, each of its member names is defined in the STOW macro instruction or in a command (as CDS) that was used when the member was added to the VPAM data set.

Programming Notes: The POD? command can be used to examine information pertaining to the members of any cataloged VPAM data set that a user owns or shares.

When the H operand is specified, the printout of system data is interpreted as:



The format of the information sent to SYSOUT after the POD? command is executed is:



Example: The user wants to obtain a listing of the modules presently in his user library.

User: pod? userlib,a
System: (presents module names and aliases)

VT (VAM To Tape) Command

This command copies a VAM data set to magnetic tape as a physical sequential data set. Used with the TV(TAPE TO VAM) command, VT allows the user to store VAM data sets on magnetic tape and retrieve them at a later time.

Operation	Operand
VT	DSNAME1=vam data set name[,DSNAME2=tape data set name]

DSNAME1

identifies the existing VAM data set, which is to be written on magnetic tape and which must already be defined by a DDEF command in the current task, or must be cataloged.

Specified as: a fully qualified data set name.

DSNAME2

specifies the name to be assigned to the magnetic-tape copy of the data set. If not specified, this data set name must be previously specified in a DDEF command with the DDNAME operand specified as DDVTOUT.

Specified as: a fully qualified data set name; if the name is preceded immediately by an asterisk, the tape data set will not be cataloged.
System default: the data set name given in the preceding DDEF command, with the data definition name of DDVTOUT is assumed. The name will be modified to the form of

TAnnnnnn.dsname1

and used as the tape data set name. Truncation of the given name to allow the insertion of TAnnnnnn (where nnnnnn is a unique number assigned by the system to assure data set uniqueness) will be from left to right.

Functional Description: The VT command may be used to copy data sets serially on tape without issuing a new DDEF command each time. Once the user has identified the first output data set name (DSNAME2) in a DDEF command with DDNAME of DDVTOUT, VT will accept each new request, update the required control information, and copy the specified data set (DSNAME1) as the next sequential file of the existing tape. The data set written out will be cataloged, if indicated, as though a new DDEF had been issued for each data set copied.

Labels are written on the magnetic tape as specified in the user's DDEF for DDVTOUT. If the current file is to be placed on an existing tape, the labeling must be consistent with the previous contents of the tape.

For each successfully copied data set, the user will receive a message indicating the names of the input and output data sets, and file sequence and volume serial numbers used. Any failure to copy successfully will result in a diagnostic message and cancellation of the command.

Programming Notes: The DDEF command describing the DDNAME of DDVTOUT must specify PS as the data set organization (DSORG) and a nine-track tape as the residence volume under the UNIT option.

Example:

1. The user wants to write his private uncataloged VAM data set, MYDS, on a private tape volume as ABC.

   User: ddef ddvtout,ps,abc,unit=(ta,9),volume=(private); vt myds
   System: (copies MYDS on tape; the name assigned to the data set on tape is TA000001.ABC)

2. The user wants to write his public VAM data set, WN3, on a private tape volume as TAPEDS2.

   User: vt dsname1=wn3,dsname2=tapeds2
   System: (copies WN3 on tape with data set name TAPEDS2)

TV (Tape To VAM) Command

This command retrieves and writes into a VAM volume, a data set previously written on magnetic tape by the VT command.

Operation	Operand
TV	DSNAME1=tape data set name[,DSNAME2=vam data set name]

DSNAME1

identifies the existing physical sequential data set residing on a nine-track tape that is to be restored to VAM on direct-access storage. The data set must already be defined by a DDEF command in the current task or must be cataloged.

Specified as: the fully qualified name with which the data set was defined or cataloged; if, when using the VT command this name was preceded with an asterisk, this data set name must be preceded with an asterisk here.

DSNAME2

specifies the name under which the data set will be cataloged. This data set does not have to be defined in the current task unless the data set is to be restored to a private VAM volume.

Specified as: a fully qualified data set name.
System default: a name will be generated by the system in the form:

DAnnnnnn.dsname1

and used as the VAM data set name. Truncation, if required, will be performed from left to right to allow the insertion of DAnnnnnn.

Functional Description: For each successfully copied data set, the user will be informed of the names of the input and output data sets, and the file sequence and volume serial numbers used. Any failure to copy successfully will result in a diagnostic message and cancellation of the command.

Examples:

1. The user wants to retrieve data set ABC, which was written on tape with the VT command, and write the data set into a public VAM volume as XYZ.

   User: tv abc,xyz
   System: (retrieves data set ABC from tape and writes it onto public VAM storage as XYZ)

2. The user wants to restore data set ABC into a private VAM volume (MYVOL1).

   User: ddef dummy,vp,xyz,unit=(da,2314),volume=(,myvol)
   tv abc,xyz
   System: (copies ABC onto private volume MYVOL)

VV (VAM To VAM) Command

This command copies a VAM data set (or program library) in direct-access storage.

Operation	Operand
VV	DSNAME1=current data set name[,DSNAME2=new data set name]

DSNAME1

identifies the existing data set, which is to be copied, and which must be defined within the current task or must be cataloged.

Specified as: a fully qualified data set name.

DSNAME2

specifies the name to be assigned to the data set copy; if the copy is to reside on a private VAM volume, the data set name must be previously defined by a DDEF command.

Specified as: a fully qualified data set name
System default: the new data set will be named in the form

DAnnnnnn.dsname1

where nnnnnn is a unique number assigned to assure uniqueness of data set names.

Functional Description: For each successful copy, the user is informed of the input and output data set names. Any failure to copy successfully results in a diagnostic message and cancellation of the command.

Examples:

1. The user wants to copy data set XYZ into public storage.

   User: vv xyz
   System: (copies XYZ with the name DA000001.XYZ)

2. The user wants to copy data set GH2 on private VAM volume MYVOL1 and name the data set ABC.

   User: ddef dummy,vi,abc,unit=(da,2311),volume=(,myvol)
   vv gh2,abc
   System: (copies GH2 onto MYVOL1 with name ABC)

CDS (Copy Data Set) Command

This command duplicates a data set or a member of a VPAM data set.

Operation	Operand
CDS	DSNAME1 = current data set name, DSNAME2 = new data set name [,DISP = { E | S }] [,BASE = first line number,INCR = increment]

DSNAME1

identifies the data set, which is to be copied, and which must already be defined by a DDEF command within the current task, or must be cataloged.

Specified as: a fully qualified data set name, and (optionally) a member name of a VPAM data set; when specified, the member name is enclosed in parentheses and immediately follows the VPAM data set name.

DSNAME2

specifies the data set name, which is to be assigned to the copy of the data set, and which must be already defined by a DDEF command within the current task, or must be cataloged. When a new member of a VPAM data set is specified, a DDEF command is not necessary if the VPAM set is already defined.

Specified as: a fully qualified data set name, and (optionally) a member name of a VPAM data set; when specified, the member name is enclosed in parentheses and immediately follows the VPAM data set name.

DISP

specifies whether the original data set or data set member residing on direct-access storage is to be erased after it has been copied.

Specified as: E - data set to be erased
S - data set to be saved
System default: S is assumed.

Note: If the user shares, but does not own, the data set being copied, he cannot specify its erasure unless his access is unlimited; if he has read-only access, this operand will be ignored.

BASE

identifies the starting line number of the data set copy, when renumbering is desired.

Specified as: one-to-seven decimal digits; an all-0 starting line number is invalid.
System default: no renumbering occurs; INCR must also be defaulted.

INCR

designates the value by which line numbers in the data set copy are to be incremented when renumbering.

Specified as: one-to-seven decimal digits; an all-0 increment is invalid.
System default: 100, when renumbering.

Note: BASE and INCR may not be specified if the data set being copied contains regions.

Functional Description: The CDS command makes a copy of the specified data set or VPAM data set member and assigns the data set name furnished by the user. When a starting line number and increment value are included in the CDS command, the lines of the data set copy are numbered accordingly, without effecting the line numbering within the original data set.

Cautions: The CDS command is restricted to data sets on direct-access or magnetic-tape volumes. CDS cannot be used to change record formats.

When CDS is used to copy object module programs, all entry points to the object program will be lost; only the module (member) name is preserved.

Before issuing the CDS command, the user must define the data set that will result and specify its organization.

Programming Notes: The rules of CDS organization are summarized in Table 5.

Table 5. CDS Facilities and Requirements

Data Organization		Residence	Definition Requirements	
Source	Copy	Source and Copy	Source	Copy
PS	PS	On either direct-access or magnetic-tape volume	Must be cataloged, or defined by previous DDEF in current task	Must be defined by previous DDEF in current task
VI	VI	Must be stored on direct-access volume		
VS	VS			
VI	VS			
VS	VI			

Table 5. CDS Facilities and Requirements (cont'd)

<table>
<tr><th colspan="2">Data Organization</th><th>Residence</th><th colspan="2">Definition Requirements</th></tr>
<tr><th>Source</th><th>Copy</th><th>Source and Copy</th><th>Source</th><th>Copy</th></tr>
<tr><td>VS</td><td>VS or VI member of VPAM data set</td><td rowspan="2">Source data set and VPAM data set receiving member must be on direct-access volumes</td><td rowspan="2">Must be cataloged, or defined by previous DDEF in current task</td><td rowspan="2">Must be defined by previous DDEF in current task, unless new member of existing cataloged data set</td></tr>
<tr><td>VI</td><td>VS or VI member of VPAM data set</td></tr>
<tr><td rowspan="2">VS member of VPAM data set</td><td>VS</td><td rowspan="3">VPAM data set provides source and copy data set, stored on direct-access volumes</td><td rowspan="3">VPAM data set must be cataloged, or previously defined by DDEF in current task</td><td rowspan="3">Must be defined by previous DDEF in current task</td></tr>
<tr><td>VI</td></tr>
<tr><td rowspan="2">VI member of VPAM data set</td><td>VI</td></tr>
<tr><td>VS</td><td></td><td></td><td></td></tr>
<tr><td>VS member of VPAM data set</td><td>VS or VI member of VPAM data set</td><td rowspan="2">VPAM data sets stored on direct-access volumes</td><td colspan="2" rowspan="2">VPAM data sets must already be defined by DDEF within current task, or must be cataloged</td></tr>
<tr><td>VI member of VPAM data set</td><td>VS or VI member of VPAM data set</td></tr>
</table>

The copy of a member may be specified as either a new member of an existing VPAM data set that the user owns or shares with access for writing, or as a new data set. Conversely, the copy of a nonpartitioned data set may be specified as a new member of a VPAM data set.

A copy of a member of a partitioned data set may have VISAM or VSAM organization. The user should specify, in the DSORG subfield of the DCB operand field in the DDEF command, the organization that is wanted for the data set copy. If he does not, and his task is conversational, he is prompted for the DCB values; if his task is nonconversational, the data set is copied with the original data set's organization.

The user may specify a VISAM organization in the DDEF command for a data set copy, even though the original set organization is VSAM, but each record of the original data set must contain a key. Also, the user should define, in the DDEF command, key length (KEYLEN), padding (PAD), and record key displacement (RKP) values. If he does not provide these values, and his task is conversational, he will be prompted for them; if the task is nonconversational, no copy is made.

An entire partitioned data set cannot be copied with one CDS command; each member must be copied individually. Separate DDEF and CDS commands, specifying the names of the data set and the member, must be issued for each member. However, the user may copy an entire library by utilizing the linkage editor to link all members of the library, effectively forming one data set.

Examples:

1. The user wants to copy cataloged VISAM data set FIRSTL, which will be a VSAM data set named TWIN.FIRSTL. He does not want to erase the original data set.

```
User:     ddef ddnz,dsname=twin.firstl
          cds firstl,twin.firstl
System:   (copies data set)
```

2. The user wants to copy VPAM line data set member SECOND (TRY). He defines this data set and its copy with DDEF commands, specifying the copy as a nonpartitioned line data set named TRYACCT. He wants to erase the original data set.

```
User:     ddef textl,vi,dsname=tryacct
          cds second(try),tryacct,e
System:   (copies data set)
```

3. The user wants to copy line data set G00R7, and name the copy G00R7A. Its lines are to be numbered in increments of 200, starting with 1000. The original data set is not to be erased.

```
User:     ddef ddn2,vi,dsname=g00r7a
          cds g00r7,g00r7a,,1000,200
System:   (copies data set)
```

4. The user wants to copy VSAM data set SEQ.DATA as a VISAM data set named VI.DATA, using a unique man number as a key in the fourth through sixth bytes of each record. The original data set contains fixed-length records, 512 bytes long.

```
User:     ddef dd2,vi,dsname=vi.data,-
          dcb=(recfm=f,lrecl=512,rkp=3,-
          keylen=3)
          cds seq.data,vi.data
System:   (copies data set)
```

5. The user has a nine-track tape (volume serial number 000126) containing BSAM data sets and wants to copy the third file on the tape on a scratch tape whose serial number will be supplied to the system by the operator.

```
User:     ddef tapel,ps,dsname=source.run,disp=old,unit=(ta,9),-
          volume=(,000126),label=3
          ddef tape2,ps,dsname=source.copy,disp=new,unit=(ta,9),-
          volume=(private)
          cds source.run,source.copy
System:   (copies data set)
```

SECTION 2: TEXT EDITING

Text is edited by using the text-editing commands, which manipulate lines of information, either within an existing region or line data set, or as they are being entered into a region or line data set. With the text editing commands, the user can simultaneously create and edit data sets; he can correct, insert, and delete lines; he can segment a data set; and he can transfer lines from one data set to another. Also, the user can display lines of a data set at his terminal, and nullify previous changes that were made by the text editing commands.

The text editing commands and the system functions they request are shown in Table 6.

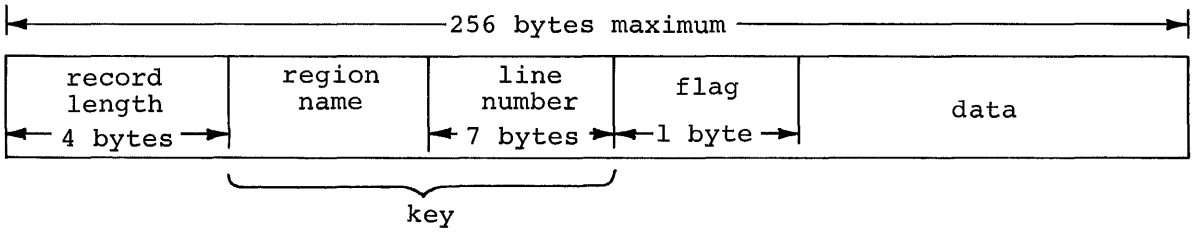
Table 6. Test Editing Commands and Their Functions (The commands are listed in the same order as the command descriptions that follow later in this section.)

Command	Function
EDIT	Invoke facilities of text editor; this command must precede the other text editing commands.
END	Terminate processing by PROCDEF and/or text editor.
REGION	Create subset of specified lines of data set, to be located as an entity known as a region.
DISABLE	Remember all modifications made in a data set in order to restore original state if requested.
ENABLE	Remember only the most recent modification made in a data set in order to restore the data set to its state before the preceding command.
STET	Delete changes to data set by previous editing command or commands; restore data set to previous condition.
CONTEXT	Replace specified string of characters within line, or range of lines, with another specified character string.
CORRECT	Change or insert characters in one or more specified lines.
REVISE	Replace specified line, or group of lines, with those entered following command.
UPDATE	Add or insert lines to current region.
EXCERPT	Insert specified region, or range of lines, from the same or another data set into current data set.
EXCISE	Delete specified line or range of lines from current data set
INSERT	Insert following lines into current data set.
NUMBER	Renumber specified line, or range of lines.
LIST	Display specified line or range of lines on user's SYSOUT.
LOCATE	Search current region for specified character string.

The user should be familiar with the terms and concepts that follow, before attempting to utilize the facilities of the text editor.

Region Data Set

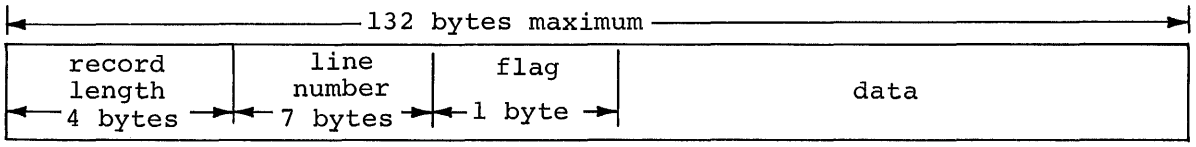
a specialized VISAM data set with this record format:



The region name is any string of 247 or fewer characters (possibly, none); the line number is a seven-digit decimal integer. The maximum record length is 256 characters. When a member of a VPAM data set is to be created and the DCB operands were not specified in the preceding DDEF command, the system-supplied DCB operand will be: LRECL=256, RECFM=V, RKP=4, and KEYLEN=15 (i.e., an eight-character region name).

Line Data Set

a specialized VISAM data set with this record format:



The line number is a seven-digit decimal integer; the maximum record length is 132 bytes. Line data sets are used mainly with the MODIFY command, but the text editor recognizes them as region data sets with zero-length regions.

Region

a line, or contiguous group of lines, whose numbers are prefixed by the same region name. The region of a data set is treated by the text editor as an entity; region names label a continuous subset of lines for identification purposes.

The size of the first specified region name in a data set will determine the size of all succeeding region names, which will be padded with blanks or truncated to fit the region field. The text editor automatically reorganizes the regions of a data set into an alphabetically ascending order, by region name.

This is a sample data set that was created by the text editor:

```
REGION          LINE NO.     Data Line

a               0000100      Text of line number 1 of REGION "a"

a               0000200      and this is next line of region "a";

a               0000300      this is end of region "a".
```

```
almost          0000100     This line starts new region, "almost"
almost          0000600     where increment has been specified as 500
almost          0001100     so line numbers advance by 500s.
nextcase        0000100     Regions in this data set can never exceed
nextcase        0000200     8 characters since creator started with
nextcase        0000300     new data set and region "nextcase."  This
nextcase        0000400     first region, specified in data set, will
nextcase        0000500     set size of region for entire data set.
nextcase        0000600     Thereafter, regions added will be truncated
nextcase        0000700     or padded with blanks to fit region fields.
nextcase        0000800     Initial allowable range of region length
ofcourse        0000100     is from 0 to 247 characters.
ofcourse        0000200     Within command system, data sets maintained
swan            0000100     by text editor, standard length of 8 has
swan            0000200     been used for regions.
```

Current Line Pointer

maintained by the text editor, the current line pointer (CLP) is set initially to 100 for empty regions and to the first line number in an existing region. The CLP is advanced through the data set as text editing commands are executed, always pointing to the next line to be processed. For further information, refer to Appendix G.

In all text editing commands, where N1 is an operand, except as indicated in the operand descriptions, the current value of the CLP may be specified by defaulting N1.

Line Number

an absolute number or a relative (plus or minus) value that indicates a number of lines away from the CLP. Example: 200 specifies line 0000200; +2 identifies the second line after the CLP; -1 designates the line preceding the CLP.

The values specified for N1 and N2 (the keyword names of the line number operands) for all text editing commands in which both operands appear are resolved by the system according to these rules:

1. When N1 and/or N2 are in the range of line numbers in current region, the system assumes the specified values.

2. If N1 is larger than the last line in the current region and/or N2 is less than the first line in the current region, a diagnostic message is issued.

3. If N1 and/or N2 fall within the range of lines in the current region, but that line number does not exist, the system assumes the next higher line number for N1 and the next lower line number for N2.

4. If N1 is not an existing line number in the current region and N2 is the same as N1, a diagnostic message results.

String Constants

are either normal or quoted. A normal string is a contiguous group of characters that begins with any nonblank character, except an apostrophe, and ends with the last nonblank character prior to either a comma, equal sign, or semicolon that is external to all pairs of parentheses in the string. A normal string may also end with the last nonblank character prior to the end of a line that does not have a continuation character. For normal strings, all System/360 characters are valid, except a comma, equal sign, or semicolon that is external to all pairs of parentheses in the string. Example:

```
A + B; (C,D), A'BC''D', C=D,A B
```

contains these normal strings:

```
A+B
(C,D)
A'BC''D'
C
D
A B
```

A quoted string is any character string that is enclosed in apostrophes, and within which all other apostrophes are doubled. All System/360 characters are valid. The internal representation of a quoted string (i.e., after it is processed by the system) does not have the terminal apostrophes; doubled apostrophes are replaced by single apostrophes. Example:

External Representation	Internal Representation
'$3.80'	$3.80
'HOW ARE YOU'	HOW ARE YOU
'I''M FINE'	I'M FINE

Hexadecimal Constant

has the form of an X followed by a string that is enclosed within apostrophes; the characters in the quoted string must be a digit or A, B, C, D, E, or F. Some examples are:

```
X'01'

X'ABC02'
```

A null character has the form of hexadecimal 0s; a null string has the form of a zero-length character string.

Break Characters

when the underscore (or any other break character selected by the user) is the first character of a line, it ensures that the statement starting on that line will be interpreted immediately as a command statement. However, when the first- and -second characters of the line are break characters, the usual break-character action will not take place. Instead, the system replaces the pair of break characters with a single break character, and processes the line as if no break character had been seen. Thus, lines starting with break

characters can be put into procedures or data sets. Use of break characters enables the user to enter commands when the system expects data.

When the vertical bar (or any other character selected by the user) is the first character of a line, it ensures that the statement starting on this line will be interpreted immediately as a language processor control statement. The rules for break-character duplication apply.

Normal Command

a command that is expected by the system and is not preceded by a break character.

Examples:

1. One of the most important applications of the text editor is to create and edit data sets. This capability is illustrated by:

User:	ddef ddname=ddl,dsorg=vi,- dsname=dsample,dcb=- (keylen=12, rkp=4, recfm=- v, lrecl=256)	(User defines new data set; key length of 12 indicates five-character region name since line numbers are seven characters)
Sys,User:	edit ddl region rname=first	(He invokes text editor; system responds with underscore; user specifies region name; text editor invites him to enter line by issuing line number)
	0000100 these are lines 0000200 of sample data 0000300 for dsample	(User enters data; each time he presses carriage return, text editor prompts with next line number)
	0000400 _update	(User wants to make change in previous entries; by preceding UPDATE with break character (underscore) text editor immediately executes command)
User:	150 new line	(User wants line 150 inserted between lines 100 and 200)
Sys,User:	_insert 400 0000400 last line	(User now wants to continue entering data at point where he left off; INSERT is preceded by underscore, since system expects data, not command, following UPDATE)
User:	_end	(Terminates text editor processing; END is preceded by underscore since system expects data, and not command after INSERT)

2. The user wants to create a nonconversational SYSIN data set that, when executed as a nonconversational task, will create a data set. He uses nested editing statements.

```
User:       ddef ddname=dcat,dsname=-     (First EDIT invokes text edi-
            boo,dcb=(keylen=12, rkp=-      tor; REGION prompts user to
            4,recfm=v,lrecl=256)           enter data lines)
Sys,User:   edit dcat
            region erb

Sys,User:   0000100 logon p3946           (Lines 100 through 900 are
            0000200 ddef ddxc, vp, jsb    text or region ERB; lines 300,
            0000300 edit ddxc, mem2       400, and 800 are nested edit-
            0000400 region onlyone        ing command statements--they
            0000500 sample data 1         are entered as data here, but
            0000600 data line 2           when SYSIN data set is execu-
            0000700 last sample data      ted, they will be executed)

            0000800_ _end                 (END in line 800 is preceded
                                          by two underscores; one will
                                          be removed when END is ac-
                                          cepted as data; when executed,
                                          END will terminate processing
                                          of nested editing statements)

            0000900 logoff

            0001000 _end                  (END here terminates process-
                                          ing of current editing state-
                                          ments)
```

When the nonconversational task is executed, member MEM2 of data set JSB will be created.

EDIT Command

This command invokes the facilities of the text editor.

Operation	Operand
EDIT	[SOURCE=data definition name] [,MNAME=member name]

Note: The user may omit either operand, but he cannot omit both.

SOURCE

identifies the data set definition associated with the data set which is to be edited.

Specified as: the data definition name specified in the previous DDEF command for the data set to be edited.
System default: SYSULIB (i.e., the data definition name of USERLIB) is assumed.

MNAME

identifies a member of a VPAM data set or library (referenced by the SOURCE operand). If the member is part of USERLIB, no prior DDEF command is necessary, and the SOURCE operand is the data definition name of USERLIB.

Specified as: the name of the member to be edited.
System default: data set is not member of a VPAM data set.

Functional Description: EDIT acts as a CALL command to the text editor. When SOURCE is omitted, the user must designate in MNAME which member of USERLIB he wants to edit; EDIT will access this member. Since it is difficult for the user to modify the existing DDEF for USERLIB, the text editor generates these DCB operand values: KEYLEN=15 (i.e., an eight-character region name), RKP=4, RECFM=V, LRECL=256.

Example: The user has defined a data set, DATAMY, that he wants to create using the text editor. The data definition name is DDNU:

User: edit ddnu

System: (informs user that text editor has been invoked and gives CLP value)

END Command

This command terminates processing of the text editor or PROCDEF command.

Operation	Operand
END	

Note: There are no operands.

Functional Description: END denotes the completion of editing initiated by an EDIT command, or terminates execution of the PROCDEF command. The command is not mandatory in either case but serves to close the data set (the user-written command is considered a data set) and to relinquish control to the associated processor.

Programming Notes: Separate EDIT and END commands as well as intervening editing commands must be issued for each data set edited.

This is not the END statement for the assembler or the FORTRAN compiler. This command may be preceded by an underscore, depending on whether it is an unexpected or an expected (i.e., normal) command; a command is unexpected when the system expects data (e.g., following an INSERT command).

Examples:

1. The user wants to terminate this editing procedure:

```
edit myprog
context 700,900,lm,stm
number 200,last
```

After execution of the NUMBER command, he enters

```
end
```

Note: END, here, is a normal command (i.e., it is expected by the system) and is not preceded by an underscore.

2. The user creates this data set.

```
User:       edit mydd
            region one

Sys,User:   0000100 line one
            0000200 line two
            0000300 _end
```

Note: END, here, is not expected; so it is preceded by an underscore.

REGION Command

This command prefixes a region name to a line number or series of line numbers, designating the line or lines as an entity. REGION also positions the data set so that editing commands can be entered.

Operation	Operand
REGION	[RNAME=region]

RNAME

identifies an existing region or specifies the region name to be assigned to a line or range of lines. The size of the first region name specified in a data set will determine the size of all succeeding region names, which will be padded with blanks or truncated to fit that region field.

Specified as: an existing region name or as a string of one to 247 characters.
System default: a null (blank) string is assumed.

Functional Description: When the user enters a region name that does not exist in the data set, the system responds with line number 100, inviting the user to enter data. The lines of data entered are prefixed by the system with the specified region name.

When the user specifies a region name that already exists, the system positions the data set to a line number 100 higher than the last line in the specified region.

When REGION is not specified and the data set contains multiple regions, specifying a line number will result in a diagnostic message.

Caution: The system automatically reorganizes the regions of a data set into an alphabetically ascending order.

Programming Notes: If editing commands with N1 and/or N2 operands are not preceded by a REGION command, the system will assume the current region name, if one exists, or a blank region name. After entering REGION, the user can reference any line in the region or enter new lines in the region by giving only the numeric line number for the line.

When an entire data set contains only one region, REGION is not necessary for entering editing statements.

To create a data set with a region name of length n, the user must precede the REGION command with a DDEF command. The region name and the line number compose the VISAM key for the record, so the user must specify a value of n+7 for the KEYLEN operand. RKP must be specified as a 4 and RECFM as V. The LRECL operand should be specified as a value

large enough to include the key length and data (maximum for LRECL is 256 characters). Example: create a data set with a region name of 35 characters.

```
ddef  ddreg,vi,dsname=xyz,dcb=(keylen=43,rkp=4,recfm=v,lrecl=256).
```

The user may create or edit a member of a VPAM data set without a preceding DDEF command. The DCB operands issued by the system will be

```
KEYLEN=15,RKP=4,RECFM=V,LRECL=256
```

The user is then limited to an eight-character region name.

Examples:

1. A region XYZ has been created with lines 100 to 500 in increments of 100. The user wants to position the data set so that he can enter editing commands.

   ```
   User:     region xyz
   System:   0000600
   ```

   The system positions the data set at XYZ600 and awaits either a new data line or reference elsewhere in region XYZ by use of the N1 or N2 operands of another editing command.

2. This is a sample data set created with text editing commands.

Region	Line No.	Data Line
a	0000100	Text of line number one of region 'A'
a	0000200	and this is next line of region 'A'.
a	0000300	This is last line of region 'A'.
next	0000100	This line starts new region called NEXT.
next	0000300	Its line increments are specified as 200,
next	0000500	so line numbers advance by 200.
next	0000700	This ends region "next."

3. Create a data set, with a region name of nine characters.

   ```
   User:     ddef dd1, vi,dsdat,dcb=(keylen=16,rkp=4,recfm=v,lrecl=256)
             edit dd1
   System:   (indicates position of CLP)

   User:     region rname=firstname
   System:   0000100     (user can now enter data)
   ```

4. Generate a region with a blank name in an existing data set.

   ```
   User:     edit mydd
   System:   (indicates position of CLP)

   User:     region
   System:   0000100
   ```

DISABLE, ENABLE, and STET Commands

These commands eliminate or make permanent previous changes to a data set.

Operation	Operand
DISABLE	

Operation	Operand
ENABLE	

Operation	Operand
STET	

Note: There are no operands in these commands.

Functional Description: The text editor maintains a transaction table where changes to a data set are recorded. Additions to the data set are noted in one column; deletions in the other. When a text editing command alters data, the change is made to the data set, and a record of that change is entered in the table. Editing commands other than ENABLE, DISABLE, STET, LIST, and LOCATE (which do not alter data) result in changes to a data set and entries in the transaction table.

The ENABLE command causes the text editor to remove previous table entries when a new command is executed; only the effect of the last-issued text editing command is present in the table while the text editor is enabled. When it is disabled (i.e., as the result of a DISABLE command), previous table entries are not removed as each new command is executed; records of all changes to a data set after DISABLE (with no intervening ENABLE) remain in the table.

Changes to a data set that are listed in the transaction table can be canceled. The STET command reverses the transaction table entries (i.e., additions are placed in the deletions column; similarly, deletions are placed in the additions column); and, at the same time, restores the data set to the status which is now indicated by the transaction table. When the text editor is disabled, all changes to a data set since the DISABLE command was issued will be in the table and will be canceled if STET is entered. When the text editor is enabled, only the last change is reversible (since it is the only change listed in the table).

When invoked, the text editor is enabled and will be until a DISABLE command is issued. The disabled text editor is enabled when the ENABLE command is entered. Execution of STET does not affect either the disabled or enabled state of the text editor. The CLP is not changed by DISABLE, ENABLE, or STET.

Cautions: The ENABLE command does not affect the transaction table until after the next text editing command that alters data. When the text editor has been disabled and ENABLE is issued, the table entries for the commands subsequent to DISABLE are not removed until after the first table entry following the ENABLE command. Thus, STET immediately following ENABLE may cause undesired results.

When multiple changes are made to a line by more than one editing command, only the last change is reversible.

Programming Notes: For efficient use of the STET command, DISABLE and ENABLE allow the user to enter all or part of his revisions before they are made permanent. When the text editor is enabled, only the last data-set change is not permanent; when disabled, all changes are temporary; they can be removed with STET.

Examples:

1. The user issues a series of commands.

```
edit  myprog
region  abc
 number  100,600
disable
excise  620
excerpt  yourprog,rname=pgr,200,400
context  700,900,joe,bob
enable
number  100,last
list
end
```

   In this series of commands, the effects of the EXCISE, EXCERPT, and CONTEXT commands are entered in the data set, and a record of these changes are collected in the transaction table. These changes become permanent when the NUMBER command, which follows ENABLE, is executed, since the entry for NUMBER in the table replaces the previous entries.

2. In example 1, assume a STET command is given between the CONTEXT command and the ENABLE command. STET cancels the effects of EXCISE, EXCERPT, and CONTEXT, since their effects were listed in the transaction table.

3. A STET command appears between the second NUMBER command and the LIST command. Since the text editor is enabled, only the effect of the NUMBER command is canceled.

4. A STET command appears between the EXCERPT and CONTEXT commands. The effects of the commands between DISABLE and STET are canceled.

5. A STET command appears between the LIST and END commands. The effect of the second NUMBER command is canceled, since the execution of LIST did not result in a transaction table entry.

6. A STET command appears between ENABLE and the second NUMBER command. STET cancels the effects of EXCISE, EXCERPT, and CONTEXT.
   Note: Although the text editor was enabled, the transaction table entries for EXCISE, EXCERPT, and CONTEXT remained; they will be removed when a table entry, following ENABLE, is made.

CONTEXT Command

This command replaces a string of characters within a line, range of lines, or region of a region data set with another character string.

Operation	Operand
CONTEXT	[N1= starting position] [,N2=ending position], STRING1=search string [,STRING2=replacement string]

N1

identifies the line or first of a range of lines in the current region to be searched for STRING1.

Specified as: a one- to seven-decimal-digit line number that may be absolute or relative.

LAST -- last line in the current region.

Note: When the user wants to start the search at a character position other than the first (position 0) in the specified line, he can specify the starting position as an absolute one-to-four digit decimal number, enclosed in parentheses and immediately following the line number.

System default: when N2 is specified, the value of the CLP is assumed; otherwise, the entire data is searched from the beginning.

N2

identifies the last of a range of lines in the current region to be searched for STRING1.

Specified as: a one- to seven-decimal-digit line number that may be absolute or relative.

LAST -- last line in the current region.

Note: When the user wants to end the search at any character position other than the last in the specified line, he can specify the ending position as an absolute one-to-four digit decimal number, enclosed in parentheses and immediately following the line number.

System default: when N1 is specified, it is the only line searched; otherwise, the entire data is searched from the beginning.

STRING1

designates the character string that is to be searched for within the range N1 to N2 (i.e., the "search argument"). Any occurrence of STRING1 that crosses a logical line boundary is not identified.

Specified as: a normal or quoted string.

STRING2

designates the character string that is to replace all occurrences of STRING1 in the range N1 to N2.

Specified as: a normal or quoted string.
System default: each occurrence of STRING1 is deleted.

Functional Description: Whenever STRING1 is found, the system replaces it with STRING2. STRING1 and STRING2 need not be the same length. If the replacement string is longer than the search string, the line will be extended to make room for the replacement string; if the replacement string is shorter, the line will be processed so that no extra spaces remain in the line after the command is executed.

If STRING1 is not specified, the user is prompted and the command is ignored. If STRING1 and STRING2 are enclosed in apostrophes (i.e., quoted strings), the apostrophes are stripped before execution of CONTEXT.

After execution, the CLP is set to the line following the last line processed or the next line location within the region.

Programming Notes: One use of the CONTEXT command is for symbol replacement in source language modules; STRING1 is the original symbol and STRING2 is its replacement. This command can be used for any source language data set, if the data set is formatted for the text editor (i.e., region or line data set).

Examples: A data set contains 20 lines, numbered in increments of 100, from 100 through 2000. The user wants to replace the string ABCDEF with UVWXYZ. Assume the CLP is 1000.

1. The user knows only that ABCDEF exists somewhere in the data set.

   User: context ,,abcdef,uvwxyz
   System: (searches entire data set, replacing ABCDEF with UVWXYZ)

2. The user wants to replace occurrences of ABCDEF that appear only in lines 500 through 1000.

   User: context 500,1000,abcdef,uvwxyz
   System: (searches lines 500 through 1000, replacing ABCDEF with UVWXYZ)

3. The user wants to replace ABCDEF only in the first 50 character positions of the line 1200.

   User: context +2,+2(50),abcdef,uvwxyz
   System: (searches first 50 characters of line 1200 and replaces ABCDEF with UVWXYZ)

4. The user wants to delete ABCDEF; he specifies an explicit null string.

   User: context -3,last,abcdef
   System: (searches lines 700 through 2000 and deletes ABCDEF)

5. The user wants to replace the quoted string 'JOHN"S HOUSE' with another string.

   User: context 0,last,'john"s house','*!@/#$'
   System: (searches entire region and replaces JOHN'S HOUSE with *!@/#$)

CORRECT Command

This command changes or inserts characters in one or more lines of the current region. The corrections are made as indicated by the lines that follow this command from the terminal. CORRECT defines the format and range of correction which is performed by a left to right scan.

Operation	Operand
CORRECT	[N1=starting line] [,N2=ending line] [,SCOL=first column] [,*$@%=replacement correction characters]

N1

identifies the line or first of a range of lines to be corrected.

Specified as: a one- to seven-decimal-digit line number that may be absolute or relative.

LAST -- last line in the current region.

System default: the current line location within the region is assumed.

N2

identifies the line or last of a range of lines to be corrected.

Specified as: a one- to seven-decimal-digit line number that may be absolute or relative.

LAST -- last line in the current region.

System default: N1 is assumed if specified; otherwise, the current line location within the region is assumed.

SCOL

specifies the character position within the lines (excluding line number), from N1 to N2, at which correction is to begin. All characters to the left of this position are ignored. The line to be corrected is displayed, starting at the SCOL position, and if the logical line length exceeds the physical line length capacity of the output terminal, only the physical line containing the character specified by SCOL is displayed.

Note: The first character position after the line number is position 0.

Specified as: one-to-four decimal digits.

System default: the first character position in the line (following line number) is assumed (i.e., position 0).

*$@%

identifies the correction characters, which will be followed by correction input from the terminal. The standard correction characters (*$@%) are replaced from the left by a direct substitution.

Specified as: a normal or quoted string.

System default: the standard correction characters are assumed.

The CORRECT command displays the designated line that is to be corrected. The standard corection characters, or the corresponding replacement correction characters, and their functions are:

* -- duplicates the character directly above the * and also all characters to the right of that first character -- until the next correction character or end of line is encountered.

$ -- duplicates the character directly above the $; all other characters on the correction line replace the corresponding characters on the original line -- until the next correction character or end of line is encountered.

@ -- duplicates the character directly above the @; all other characters in the corresponding fields in the replacement line (which follows the correction line) replace the corresponding characters in the

original line -- until the next correction character or end of line is encountered. The fields in the replacement line are terminated by @.

% -- removes the character directly above the %, i.e., all positions to the right are shifted left one position; all other characters to the right of that first character are duplicated -- until the next correction character or end of line is encountered.

Functional Description: The system displays the line to be corrected and the correction is specified by the next line (i.e., correction line) input from the terminal after issuance of the CORRECT command. The particular operations desired are indicated by the use of the correction characters. Until a correction character is detected, it is assumed that all nonblank characters in the correction line replace the corresponding characters in the line to be corrected. The replacement line, if required, follows the correction line from the terminal.

The user is prompted if the replacement line ends before the replacement field is flagged by the @ character. Correction continues according to the correction-character line.

An end of line in a field marked by * or % causes the remainder of the line to be duplicated; an end of line in a field marked by $ terminates the line.

Examples:

1. User: correct 400
   System: STEMS3660 (displays line to be corrected)

   User: @* $ *%*
   ys@

   (SYSTEM 360 is resultant line)

   In example 1, the first @ in the correction line caused the YS from the replacement line to be inserted; the $ followed by a blank caused the blank to replace the character above it (S); and the % caused the character above it to be deleted and all following characters to be shifted to the left.

2. User: correct 104
   System: COMPVTE X1 (Displays line to be corrected)

   User: * $U* (Correction line; this replaces V in line to be corrected with U from the correction line, and duplicates all following characters)

   (COMPUTE X1 is resultant line)

3. User: correct 15, 16, pgrs (pqrs replaces standard correction characters)

   System: L 1 4 , X (Displays lines to be corrected)
   S T 1 4 , Y (These are not automatically displayed, but are shown here for clarity.)

   User: P Q 5S (Correction line; duplicates to blank before 14, replaces 1 by 5, deletes 4, and shifts X and Y one space to left.)

   (L 5,X and ST 5,Y are resultant lines)

4. User: correct 27

   System: CONTNUE (Displays line to be corrected)

   User: * @* (Correction line; * duplicates through CONT; @ indicates that field from next line is to be inserted after CONT and before N; * duplicates remainder of line from N.)

   i@ (Replacement line; I is value of replacement field)

   (CONTINUE is resultant line)

5. User: correct 15

   System: xyz 1345 co mpte x (Displays line to be corrected)

   User: abc @ * %@ *
   L3, 4, 5@ u@ (Correction and replacement lines; XYZ is replaced by ABC automatically, 1345 is replaced by L3, 4, 5 from following line; space between O and M is deleted; U is inserted from following line)

   (ABC L 3,4,5 COMPUTE X is resultant line)

REVISE Command

This command specifies the point or range in a data set at which the lines following this command may be inserted.

Operation	Operand
REVISE	[N1=starting line] [,N2=ending line [,INCR=increment]

N1

identifies the line or first of a range of lines to be referenced, or deleted and subsequently replaced.

Specified as: a one- to seven-decimal-digit line number that may be absolute or relative.

LAST -- last line in the current region.

System default: the value of the CLP is assumed.

N2

identifies the last in a range of lines to be deleted and subsequently replaced.

Specified as: a one- to seven-decimal-digit line number that may be absolute or relative.

LAST -- last line in the current region.

System default: the value of N1 is assumed.

INCR

designates the increment of the newly assigned line numbers.

Specified as: one-to-seven decimal digits; an all-zero increment is not valid.
System default: 100 is assumed.

Functional Description: When REVISE is not followed immediately by EXCERPT or a data line or lines, the command deletes the specified lines and sets the CLP to N1. When REVISE is followed by EXCERPT or data lines, all lines specified by REVISE are deleted and the new lines are inserted and line numbers are assigned; the CLP is set to the number of the last data line entered plus the current increment. REVISE deletes the specified lines first, then inserts the data lines.

The user is prompted if the number of insertion lines exceeds the upper limit specified by N2.

Programming Notes: REVISE is functionally equivalent to an EXCISE command followed by an INSERT or EXCERPT command.

Examples:

1. Replace line 300 with data lines in increments of 10.

   User: revise 300, incr=10
   System: (deletes line 300 and waits for user to enter data lines or next command)

2. Replace lines 200 through 550 with data lines in increments of 100.

   User: revise 200, 550
   System: (deletes lines 200 through 550 and waits for user to enter data lines or next command)

   Note: The increment of 100 is the default value of INCR. If more than four lines are inserted, the fifth line number will exceed the N2 value (assuming the next higher line number, if 600 or less) and cause a diagnostic message to be displayed before insertion.

UPDATE Command

This command adds or inserts the data lines entered at the terminal into the current region.

Operation	Operand
UPDATE	

Note: There are no operands.

Functional description: The lines of data entered by the user following this command are inserted in the current region at the specified line number. If the user specifies a line number that already exists in the region, a message is displayed; the new line will overlay the old line.

UPDATE is terminated when a text editing command preceded by an underscore is issued. The status of the CLP does not change during execution of UPDATE.

Caution: When issuing insertion lines, a space must appear between the line number and the text of the line.

Programming Notes: UPDATE is equivalent to a series of INSERT or REVISE commands. UPDATE is intended primarily to allow the insertion of arbitrary line numbers; INSERT and REVISE are designed for consecutive line insertions.

Example: Assume the current line location is in the region ABC, which contains 10 lines, numbered 100 through 1000 in increments of 100. The user wants to insert a line between lines 200 and 300 and one between lines 600 and 700. He also wants to replace line 500 with a new line.

```
User:     update
System:   (prompts user to enter data line)

User:     250 datal
System:   (inserts line 250 between 200 and 300; prompts user to
            enter data line)

User:     650 text
System:   (inserts line 650 between 600 and 700; prompts user to
            enter data line)

User:     500 more data
System:   (replaces old line 500 with new line and issues message;
            prompts user to enter data line)

User:      insert 1100
System:   (terminates execution of UPDATE, positions CLP to line
            1100)  0001100

User:     (enters data line)
```

EXCERPT Command

This command inserts a region or range of lines from another data set into the current data set. EXCERPT, following a REVISE command, replaces a range of lines in the current data set; following an INSERT command, it adds to lines being typed in from the terminal.

Operation	Operand
EXCERPT	DDNAME = data definition name [.member name] [,RNAME = region name] [,N1 = starting line[,N2 = ending line]]

DDNAME

identifies the data set definition of the data set from which the region, line, or range of lines is to be taken. This data set must already be defined by a DDEF command within the current task. When DDNAME refers to a VPAM data set, a member name must also be specified.

Specified as: the data definition name, specified in a previous DDEF command, of the referenced data set; when a member of a VPAM data set is referenced, the member name is preceded by a period and immediately follows the data definition name.

RNAME

identifies the region, previously defined in a REGION command, from which data is to be inserted into the current data set, either in its entirety, or within the range specified by N1 and N2.

Specified as: the name of the region, expressed as a normal or quoted string, from which data is to be copied.

System default: when N1 and N2 are specified, it is assumed that the data set named in DDNAME contains only one region; when N1 and N2 are both omitted, the entire data set named in DDNAME is inserted as a single region; when only N2 is omitted, the line designated by N1 is inserted.

N1

specifies the line or the first of a range of lines that is to be inserted in the current region.

Specified as: an absolute one- to seven-decimal-digit number.
System default: N2 must also be omitted; the entire region specified in RNAME is inserted.

N2

specifies the last in a range of lines that is to be inserted in the current region.

Specified as: an absolute one- to seven-decimal-digit number.
System default: the entire data set is inserted when N1 and RNAME are omitted; when only N2 is omitted, only the line specified by N1 is inserted.

Note: This operand cannot be specified unless N1 is used.

Functional Description: Upon completion of this command, the current line pointer is unchanged. Insertion is always made immediately after the data line that was the current line location at the time the command is issued. The system automatically renumbers the inserted lines. If the existing line numbers do not accomodate the number of lines to be inserted (e.g., trying to insert more than 99 lines between line numbers 400 and 500), the command is not executed and a diagnostic message is issued.

The user is prompted when these exception conditions occur:

- Renumbered lines would overflow the interval between lines.
- The data set and region to be excerpted could not be found.
- The line number within the region could not be found.
- The end of the region was reached before any of the requested lines could be excerpted.

Examples: Assume a data set with DDNAME=ABC has four regions, ABC1, ABC2, ABC3, and ABC4; region ABC1 has lines numbered from 100 through 1000 in increments of 100.

1. Excerpt the entire data set.

   User: excerpt abc
   System: (inserts data set with data definition name ABC into current data set)

2. Excerpt only lines 300 through 500 from region ABC1.

   User: excerpt abc,abc1,n1=300,n2=500
   System: (inserts lines 300 through 500 of region ABC1 into current data set)

3. Excerpt lines from a member of a VPAM data set.

User: excerpt abc.mem,reg,99,1000
System: (inserts lines 99 through 1000 from member MEM or VPAM data set with data set definition name ABC into current data set)

EXCISE Command

This command deletes a line or a group of lines from a region.

Operation	Operand
EXCISE	[N1 = starting line] [,N2 = ending line]

N1

designates the line or first of a series of lines to be deleted from the current region.

Specified as: a one- to seven-decimal-digit line number that may be absolute or relative.

LAST -- last line in the current region.
System default: the value of the CLP is assumed.

N2

designates the last line in a series of lines to be deleted from the current region.

Specified as: a one- to seven-decimal-digit line number that may be absolute or relative.

LAST -- last line in the current region.
System default: only line specified in N1 is deleted.

Functional Description: After EXCISE is executed the CLP is set to the first line deleted or, if no lines were deleted, to the value specified in N1.

Programming Notes: Since the CLP is set to the first line deleted, or the value of N1, the user may conveniently follow this command with either an INSERT or EXCERPT command.

Examples:

1. Delete line 113.

User: excise 113
System: (deletes line 113)

2. Delete the next 10 lines beyond the current position in the data set.

User: excise n1=+1,n2=+10
System: (deletes the 10 lines beyond CLP)

3. Delete any lines between 100 and 300.

User: excise 101,299
System: (deletes all lines between 100 and 300)

INSERT Command

This command places the data lines entered at the terminal in the current region.

Operation	Operand
INSERT	[N1 = preceding line] [,INCR = increment]

N1

identifies the line that is to precede the inclusion of the following data lines in the current region of a data set or member.

Specified as: a one- to seven-decimal-digit line number that may be absolute or relative.

LAST -- last line in the current region.
System default: the value of the CLP is assumed.

INCR

specifies the value by which line numbers assigned to the new data lines are incremented. The value of N1 is the base against which the line numbers are incremented.

Specified as: one-to-seven decimal digits; an all-zero increment is not allowed.
System default: an increment of 100 is assumed.

Functional Description: If there is not adequate space between existing line numbers for the specified insertions, the user is prompted with a message. If there is adequate space, line numbering of the inserted lines is done by the system.

Examples:

1. Insert data lines following line 600 in increments of 10.

   User: insert 600,10
   System: 0000610

   Note: Assuming the line following 600 is 700, nine lines can be inserted before overflowing the space.

2. Insert lines in the data set 10 lines beyond the current position with an increment of 100.

   User: insert n1=+10
   System: (issues line number prompting user to enter data line)

3. Position the CLP following the last line or insert lines at this point.

   User: insert last
   System: (issues line number prompting user to enter data line)

4. Generate a region with a blank name in a new data set.

   User: edit mydd
   System: (indicates the position of the CLP)

   User: insert
   System: 0000100

Note: REGION must be used to generate a region with a blank name in an existing data set. EDIT automatically positions the CLP at the end of the last region in an existing data set and INSERT, without operands, assumes this value for the CLP.

NUMBER Command

This command renumbers a line or a range of lines within a region or contiguous regions.

Operation	Operand
NUMBER	[N1 = starting line] [,N2 = ending line] [,BASE = base number] [,INCR = increment]

N1

identifies the line or first of a range of lines to be renumbered.

Specified as: a one- to seven-decimal-digit line number that may be absolute or relative.

LAST -- last line in the current region.

System default: see Table 7 below.

N2

identifies the last of a range of lines to be renumbered.

Specified as: a one- to seven-decimal-digit line number that may be absolute or relative.

LAST -- last line in the current region.

System default: see Table 7 below.

BASE

indicates the number from which the renumbering is to be incremented.

Specified as: one-to-seven decimal digits; it must not be less than N1.

System default: see table 7 below.

INCR

specifies the increment between the lines to be renumbered.

Specified as: one-to-seven decimal digits; if the increment causes renumbering to overlap the number of the line following N2, the increment is computed as though it were defaulted, and the user is prompted with a message informing him that the increment has been furnished by the system.

System default: the system computes the increment as follows: the difference between the base and the line number following N2 is

divided by the number of lines to be renumbered, plus one. The increment is then determined in this manner:

If the quotient is	Then the increment will be
100 or greater	100
50-99	50
20-49	20
10-19	10
5-9	5
2-4	2
1	1

Table 7. System Defaults for NUMBER Command Operands

These are the system defaults and the results expected for the NUMBER command (an X indicates that the specific operand has been explicitly given or there is a default for it in the user's profile). Other combinations can be derived from the table.

Case	N1	N2	BASE	INCR	Comment
1	X	N1	N1	100-	Only line specified by N1 is renumbered
2	CLP	X	N1	100-	Lines from CLP to N2 are renumbered
3	CLP	CLP	X	100-	Only line specified by CLP is renumbered
4	CLP	CLP	CLP	X	Only line specified by CLP is renumbered
5	X	X	N1	100-	
6	0	LAST	100	100	If no operands are specified, entire region is renumbered

Functional Description: N1 and N2 must be numbers within the same region. Renumbering does not affect the region name prefixed to line numbers.

When all operands are defaulted, the entire data set is renumbered; renumbering will occur across region boundaries.

Upon completion of this command, the CLP is positioned to the line following N2.

If the BASE is specified as less than N1, a diagnostic message is issued. If the value of INCR causes the renumbering to overlap the line number specified in N2, the system computes the increment as if it were defaulted, and notifies the user by a message; renumbering with the new increment then occurs.

Examples:

1. number 103, 290

Original Sequence	Resulting Sequence
XYZ0000100	XYZ0000100
XYZ0000103	XYZ0000123
XYZ0000107	XYZ0000143
XYZ0000108	XYZ0000163
XYZ0000109	XYZ0000183
XYZ0000111	XYZ0000203
XYZ0000114	XYZ0000223
XYZ0000116	XYZ0000243
XYZ0000169	XYZ0000263
XYZ0000290	XYZ0000283
XYZ0000400	XYZ0000400

Since BASE is defaulted, it is assumed to be 103 (N1). The difference between the base and the line following N2 (400) is 297, which is divided by the number of lines plus one. Since the quotient is 29.7 (297 ÷ 10 = 29.7) the increment is 20.

2. number 17

Original Sequence	Resulting Sequence
AR0000010	AR0000010
AR0000017	AR0000022
AR0000035	AR0000035

3. number 912, 1000

Original Sequence	Resulting Sequence
AR0000900	AR0000900
AR0000912	AR0000932
AR0000915	AR0000952
AR0000916	AR0000972
AR0000917	AR0000992
AR0000918	AR0001012
AR0001000	AR0001032
AR0001050	AR0001050

4. number 5,12,base=6,incr=13

Original Sequence	Resulting Sequence
M0000001	M0000001
M0000005	M0000019
M0000008	M0000032
M0000009	M0000045
M0000100	M0000100

5. number 100,200

Original Sequence	Resulting Sequence
100	120
125	140
150	160
200	180
250	250

LIST Command

This command displays the value of the CLP, or a line or range of lines at the user's terminal or SYSOUT. LIST does not alter the referenced data.

Operation	Operand
LIST	[N1 = starting position] [,N2 = ending position]

N1

identifies the line or first of a range of lines in the current region to be displayed, or specifies that the value of the CLP is requested.

Specified as: one- to seven-decimal-digit line number that may be absolute or relative.

LAST -- last line in the current region.
CLP -- value of current line pointer.

Note: If the user wants to start the listing at a character position other than the first (position 0) in the specified line, he can specify the starting position as an absolute one-to-four-digit decimal number, enclosed in parentheses and immediately following the line number.

System default: when N2 is specified, the value of the CLP is assumed; otherwise, the entire data set is listed.

N2

identifies the last line in a range of lines of the current region to be displayed.

Specified as: one- to seven-decimal-digit line number that may be absolute or relative.

LAST -- last line in the current region.

Note: If the user wants to end the listing at any character position other than the last in the specified line, he can specify the ending position as an absolute one- to-four-digit decimal number, enclosed in parentheses and immediately following the line number.

System default: when N1 is specified, it is the only line listed; otherwise, the entire data set is listed.

Functional Description: After LIST is executed the CLP is set to the next line number after N2. If N2 is the last line of the data set, the CLP is set to N2+100 and the last two digits are set to 00.

Examples: The user has previously issued a REGION command to create this data set.

```
User:      region anyregn

Sys,User:  000100 line 1
           000200 line 2
           000300 line 3
```

1. Assuming the CLP is 300, the user can issue any of these LIST commands to display the entire data set.

```
User:     list  100,300
or        list -2,last
or        list -5,+1
          (Note: When N1 and N2 exceed the limits of the data set,
          the lowest and highest line numbers are assumed)
System:   ANYREGN0000100        LINE 1
          ANYREGN0000200        LINE 2
          ANYREGN0000300        LINE 3
```

2. Display all but the first two characters of line 200 and the first four characters of line 300.

```
User:     list 200(2),300(4)
System:   YREGN0000200          LINE 2
          ANYR
```

3. Display the value of the CLP.

```
User:     list CLP
System:   (displays value of CLP)
```

4. List the second character of line 100.

```
User:     list 100(1),100(2)
System:   N
```

LOCATE Command

This command searches a region for a specified character string.

Operation	Operand
LOCATE	[N1 = starting position] [,N2 = ending position [,STRING = character string]

N1

identifies a line or first of a series of lines in the current region to be searched for STRING.

Specified as: a one- to seven-decimal-digit line number that may be absolute or relative.

LAST -- last line in the current region.

Note: If the user wants to start the search at a character position other than the first (position 0) in the specified line, he can specify the starting position as an absolute one-to-four-digit decimal number, enclosed in parentheses and immediately following the line number.

System default: when N2 is specified, the value of the CLP is assumed; otherwise, the entire data set is searched.

N2

identifies the last of a series of lines to be searched for STRING.

Specified as: a one- to seven-decimal digit line number that may be absolute or relative.

LAST -- last line in the current region.

Note: If the user wants to end the search at any character position other than the last in the specified line, he can specify the ending position as an absolute one-to-four-digit decimal number, enclosed in parentheses and immediately following the line number.

System default: when N1 is specified it is the only line searched; otherwise, the entire data set is searched.

STRING

designates the character string that is to be searched for (i.e., the string is the "search argument"). Strings that cross logical line boundaries are not recognized.

Specified as: normal or quoted string.
System default: the CLP is set to the next available line number in the current region.

Functional Description: LOCATE searches the specified lines for the string; when the string is found, the first line containing it is displayed and the CLP is set to that line number; when the string is not found, or LOCATE was issued without operands, the CLP is set to the last line in the range specified in the region, current plus 100.

Examples:

1. The user has issued a previous REGION command and wants to search the entire region for the string ABC.

   User: locate ,,'abc'
   System: (displays physical line in which ABC was first found)

2. The user narrows the search to lines 200 through 500.

   User: locate 200,500,abc
   System (displays physical line in which ABC was first found)

3. The user restricts the search in Example 2, to character-position 3 of line 200 through character-position 26 of line 500.

   User: locate 200(3),500(26),abc
   System: (displays physical line in which ABC was first found)

SECTION 3: DATA EDITING

Three commands, used to build and edit VSAM and VISAM data sets, are shown in Table 8.

Table 8. Data Editing Commands and Their Functions (The commands are listed in the same order as the command descriptions that follow.)

Command	Function
DATA	Build VSAM or VISAM line data set.
MODIFY	Insert, delete, and/or replace lines in VISAM data set.
LINE?	Obtain lines from line data set or from language processor listing data set.

DATA Command

This command creates either a line data set or a VSAM data set.

Operation	Operand
DATA	DSNAME=data set name [,RTYPE=I [,BASE=first line number, INCR=increment]]

DSNAME

identifies a new data set or a new member of a partitioned data set. The data set must be defined within the current task by a DDEF command unless it is to reside on public storage.

Specified as: a fully qualified data set name, and (optionally) the member name of a VPAM data set; when specified, the member name must be enclosed in parentheses and immediately follow the VPAM data set name.

RTYPE

indicates that the data set is in line organization.

Specified as: I
System default: VSAM is assumed.

BASE

identifies the starting line number of the line data set being created.

Specified as: three-to-seven decimal digits, the last two of which should be 0s; an all-zero starting line number is invalid.
System default: 100 is assumed.

INCR

specifies the value by which the line numbers in the data set are to be incremented.

Specified as: three-to-seven decimal digits, the last two of which should be 0s; an all-zero increment in invalid.
System default: 100 is assumed.

Note: The specification of BASE and INCR is invalid for a sequential data set (indicated by defaulting RTYPE).

Functional Description: Either a line data set (the I option is selected) or a VSAM data set is created. The user can modify, correct, insert, and delete lines only in a line data set.

When the DATA command is used to insert data into public storage, the system issues the DDEF command on the user's behalf; the newly created data set is automatically cataloged for him. (But the user must issue a DDEF command if his data set is to reside on private storage.)

In conversational mode, if indexing is specified, the system requests each line by issuing the current line number; if indexing is not specified, the system prompts for each line by issuing the number sign (#). When the user has entered all of his input source data, he must indicate this to the system by entering the end record, %E.

Lines of a line data set being entered with the DATA command can be modified, corrected, or deleted, and new lines can be inserted, by following these conventions:

1. Modify or correct a line of a line data set.

   %line number, data

   line number

   identifies the line to be replaced by a modified or correct line.

   data

   the replacement line of input data.

2. Insert a new line into a line data set.

   %line number, data

   line number

   identifies the new line to be inserted. It may be any one- to seven-digit integer whose value specifies the location of the new line within the data set; this value must not exceed the last existing line number.

3. Delete a line or a series of lines from a line data set.

   %D,line number [,last line number]

   line number

   identifies the line to be deleted.

   last line number

   identifies the last line to be deleted, if a sequence of lines is being deleted, starting with "line number;" "last line number" must be higher in value than "line number."

If the ATTENTION key is pressed while a DATA command is in operation, the action taken depends on the type of data set being created. If the data set being created is VSAM, it is eliminated when the ATTENTION key is pressed. If a line data set is being created, all that has been entered, before the attention interruption, is saved. In either case, the system asks the user for his next command. The user then has the option, for a line data set only, of entering a MODIFY command and continuing to create his line data set.

Cautions: The DATA or MODIFY command names must not be included in records entered with a DATA command. The text editor does permit this type of nesting. The first %E found is interpreted as the end-of-input record for the current DATA command. Therefore no record starting with % (except %END) may be entered into a line data set.

When the user references an empty data set that was created in a previous task with DATA or the text editing commands, his current task may be abnormally terminated. This is a VAM limitation which will eventually be removed.

Programming Notes: The maximum line length is 120 characters of text (not counting line number) for either a line data set or a VSAM data set. When records are being entered via the IBM 1056 Card Reader with the AUTO EOB switch on, the maximum record length is 80 characters; with the switch off, the maximum length is 79.

When there is a continuation, the continuation character is not included in the record placed in the data set. Each continuation line (i.e., a line that continues the statement initiated in a preceding line) is accepted as if it is a new and independent line that forms a complete statement by itself.

DATA normally puts a new data set on a public volume. If a private volume is desired, a DDEF command must be issued for a data set before the DATA command is issued.

When a data set is being created to serve as SYSIN for a task, it may include data that is to be read by the user's object program at execution time. In this case, the data must follow immediately after the command that is to start user program execution. An end-of-data line, %END, must follow the last data line.

Examples:

1. The user is attempting to construct a line data set named ROVER1.

```
User:      data rover1,i,100,200

Sys, User: 100 subroutine alpha (beta)

           300 common gamma(3,5),delta(10),epsilon

           500 param=beta

           700 %350,common theta

           700 %35G,integer beta

System:    (informs user that %35G is an invalid correction number --
           line is ignored)

User:      %355,integer beta

Sys, User: 700 10 format(5x,I7)

           900 %700,10 format (5x,I8)

           900 %d,350,355

           900 do 25 i=1,3

           1100 do 25 j=1,5

           1300 %950,gamma(1,1)=param
```

```
1300 gamma (i+1,j)=gamma(i,j)*param

1500 %E
```

2. The user wants to construct a VSAM data set made up of a sequence of commands to be used in a BACK command; the data set is to be named COMSET.

```
User:      data comset
Sys, User: #ftn raader,n,,,y,y,y,y

           #logoff

           #%e
```

MODIFY Command

This command inserts, deletes, replaces, or reviews lines of a VISAM line data set or a VISAM member of a VPAM data set; or creates a VISAM data set or member.

Operation	Operand
MODIFY	SETNAME=data set name [,CONF=R] [,LRECL=record length,KEYLEN=key length RKP=key displacement,RECFM= {V|F}]

SETNAME

identifies a VISAM data set. If the data set already exists, it must have been defined previously by a DDEF command within the current task or must have been cataloged; the data set to be created by MODIFY need not be defined or cataloged.

Specified as: a fully qualified data set name, and (optionally) member of a VPAM data set; when specified, the member name is enclosed in parentheses and immediately follows the VPAM data set name.

CONF

specifies that review of modifications is requested; each line of the data set that was changed is presented to the user in its original form.

Specified as: R
System default: no review of lines.

Note: The next four operands must all be explicitly specified or all be defaulted.

LRECL

designates the length, in bytes, of each fixed-length logical record.

Specified as: a decimal number; maximum length for VISAM, 4000 bytes.
System default: line data set is assumed.

KEYLEN

designates the length, in bytes, of the key associated with each physical record. When a record is read or written, the number of bytes transmitted equals the key length plus the record length.

Specified as: a decimal number; maximum key length, 244 bytes.
System default: line data set is assumed.

RKP

specifies the displacement of the key field from the first byte of each logical record. If the record is variable length, the first byte of the record is at RKP=4.

Specified as: a decimal number.
System default: 0 is assumed.

RECFM

indicates the format of the data set records.

Specified as: V - variable-length records
F - fixed-length records.
System default: V is assumed.

Functional Description: The system asks the user for the line modifications by issuing a number sign (#). The user indicates his modifications by following these conventions:

1. Insert or replace a line of data.

   line,data

   line

   the line number or key of the line of data to be inserted or replaced.

   Specified as: one-to-seven digits.

   data

   the new data of the replacement or insertion line; a maximum of 120 characters is permitted in a line data set.

2. Delete a line or a range of lines.

   D,line [,last line]

   line

   the line number (or key) of a single line to be deleted or the first line number (or key) of a range of lines to be deleted.

   last line

   the final line number (or key) of a range of lines to be deleted.

3. Review a line or range of lines (whether or not the review option is specified) without taking any other action with the lines.

   R,line [,last line]

   line

   the line number (or key) of a single line to be reviewed or the first line number (or key) of a range of lines for review.

   last line

   the final line number (or key) of a range of lines to be reviewed.

   The user indicates that he has completed his modifications by entering %E.

When the review option is requested, the line deleted or replaced, is presented after each modification.

If the ATTENTION key is pressed while the MODIFY command is in operation, it does not affect the modifications that have been entered up to the moment of interruption. Those modifications are made in the user's data set. The MODIFY command operation is terminated, however, and the system requests the user's next command. If desired, he may then enter a new MODIFY command and continue making modifications to his data set.

The MODIFY command will accept strings of EBCDIC representations of hexadecimal digits, convert them into machine representations of hexadecimal digits, and insert them in a data set as directed by the user.

The EBCDIC string representing the hexadecimal data is entered in this format:

X%EBCDIC string (any non-EBCDIC character ends hexadecimal data)

When the system encounters the X and the immediately following %, it enters hexadecimal mode. It then assumes that an EBCDIC string follows and proceeds to convert each character in the string to a hexadecimal digit, until the first nonhexadecimal character is encountered.

In performing the required conversion, the system checks that each input character represents a valid hexadecimal digit; that there is an even number of input characters in the string; and, that there are no incomplete inserts in any input line (more than one insert may be made in any input line; however, one insert may not be entered across input lines).

When characters, not in hexadecimal format, are written at the terminal, they will be lost in the transmission and will not be marked, even by a space.

Caution: The DATA or MODIFY command names should not be included in the records entered under a MODIFY command. The first %E is interpreted as the end-of-input record for the current MODIFY command.

Programming Notes: The user, to save processing time, should enter his modifications in sequence, starting with the lowest line number.

By making a series of insertions, the user can use the MODIFY command to create a new VISAM data set.

When a data set that is to serve as SYSIN is being built from records entered via the card reader, the maximum record length must be 80 characters. In this case, continuation conventions must agree with those specified for card input (see Part II, "Entering Command Statements").

When the data set being created or modified is to serve as SYSIN for a task, it may include data that is to be read by the user's object program at execution time. In this case, the data must follow immediately after the command that is to start the user's program execution. An end-of-data line with %END must follow the last data line.

The user may create a VISAM data set, other than a line data set, that includes his own keys. If so, he must give the key position and length within the record. These key values may then be used to insert, replace, delete, or review lines while the data set is being built: For example, if the user enters

```
AB14000 link, upper arm
```

he must have previously specified, in the MODIFY command, KEYLEN=5 and RKP=3. Thus 14000 is the indexing key to his record.

When creating a record that is longer than one line, the user must enter a hyphen at the end of the line to signal that the next line is a continuation. The hyphen does not become part of the record itself; the continuation line is not prefixed with a key.

Note: MODIFY, although much less flexible than the text editing commands, does permit use of a VISAM key anywhere in the record; the text editor works only with line and region data sets (i.e., key in first position).

Examples:

1. The user wants to delete lines 107 through 195 and replace line 107 in data set ASET. Review is not requested.

```
User:       modify aset
System:     (requests modifications)

Sys, User:  #d,107,195

            #107,x=a**2.0

            #%E

System:     (verifies that modification is complete)
```

2. The user wants to delete line 4900 and insert a new line at line number 5450 in his partitioned data set AB12.CA(V8). He requests review.

```
User:       modify ab12.ca(v8),r
System:     (requests modifications)

Sys,User:   #d,4900
System:     4900,X=(X=C) (reviews deleted line)

Sys,User:   #5450,j=j+1

System:     5400,K=J(reviews line that precedes insertion)

Sys,User:   #%e

System:     (verifies that modification is complete)
```

3. The user wants to replace line 12300 and insert a new line at 14350 in his data set DAT.C. He requests review.

```
User:       modify dat.c,r
System:     (requests modifications)

Sys,User:   #12300,somer=b/c
System:     12300,SOMER=B-C (reviews old line)

Sys,User:   #14350,i=12
System:     14300,PARAMB=0 (reviews line that precedes insertion)

Sys,User:   #%e
System:     (verifies that modification is complete)
```

4. The user wants to create a new non-line VISAM data set named QUIK4. Records are to be 80 bytes, fixed-length; the key is a five-digit part number, displaced two characters from the start of the record. Review is not wanted.

```
User:       modify quik4,lrecl=80,keylen=5,rkp=3,recfm=f
System:     (requests modifications)

Sys,User:   #ab00411 spring,retaining

            #ab00412 spring,guide

            #ab00413 clip,retaining spring

            #ab00414 widget,silverplated

            #%e

System:     (verifies that modification is complete)
```

5. The user wants to create a new line data set, named DISSMAL. Review is not wanted.

```
User:       modify dissmal
System:     (requests modification and informs user that new data
            set will be created)

Sys,User:   #100,ald dc f'875'

            #200,smel dc f'5280'

            #300,dc f'6793'

            #400, dc f'557'

            #%e

System:     (verifies that modification is complete)
```

LINE? Command

This command presents one or more lines from a line data set to SYSOUT. A maximum of ten line-number ranges may be specified in a single execution.

Operation	Operand
LINE?	DSNAME=data set name [{line number (first line number, last line number)} [,...]]

Note: Only DSNAME may be specified in keyword format.

DSNAME

identifies a line data set, which must be defined by a DDEF command within the current task, or must be cataloged, and from which a line is to be displayed.

Specified as: a fully qualified data set name.

line number

identifies a single line to be displayed.

Specified as: a one- to seven-decimal-digit number.
System default: if first line number, last line number operand is specified, that range of lines is displayed; otherwise, entire contents of data set are displayed.

first line number, last line number

identifies a range of lines to be displayed.

Specified as: two one- to seven-decimal-digit numbers, separated by a comma, and enclosed in parentheses.
System default: If line number operand is specified, that line is displayed; otherwise, the entire contents of the data set are displayed.

Functional Description: When the user specifies a line number or a beginning-of-range line number that does not exist but is within the bounds of the data set, the next higher line is presented.

If the line number is greater than the highest (or lower than the lowest) line number in the data set, the user is informed of the highest (lowest) line.

If the user specifies a range of line numbers that in some way overlaps the boundaries of the data set, all lines in the data set within the specified range are presented. If the range overlaps the end of the data set, the user is informed when the end of the data set is reached.

Format of output for line data set:

Byte Position	Content
1-7	line number
8	blank if line was created from terminal keyboard; C if line was created from card reader
9	text

Format of output for language processor listing data set:

Byte Position	Content
1-130	text (record positions 2 through 131)

Caution: In the specification of a range of line numbers, the beginning-of-range line number must be less than or equal to the end-of-range line number.

Programming Notes: In conversational mode, the user can terminate the presentation at any point by pressing his ATTENTION key.

The user can present lines only from a data set that belongs to him or that he is now sharing. He may request the lines in any numerical sequence.

Examples:

1. The user wants lines 800 through 1100 and line 1400 of data set NAM3 to be presented.

   User: line? nam3,(800,1100),1400
   System: (presents lines 800-1100 and 1400)

2. The user want lines 900 through 2400 and lines 4400 through 16000 of member AB1 of VPAM data set REPLAY to be presented.

   User: line? replay(ab1),(900,2400),(4400,16000)
   System: (presents lines 900-2400 and 4400-16000)

3. The user wants his entire data set, LIST.PLAYER, to be presented.

   User: line? list.player
   System: (presents contents or entire data set)

SECTION 4: BULK OUTPUT

The bulk output commands, given in Table 9, allow the user to transfer data sets from his virtual storage to output devices other than the terminal he uses in conversational mode. The output printer, at the central computer installation, can write data sets more rapidly than at the user's terminal. WT and PUNCH put data sets on cards and tape, which are not available at the user's terminal. Each command in this group initiates action in a nonconversational task to accomplish the data transfer, thereby freeing the user from the need to monitor bulk output.

Table 9. Bulk Output Commands and Their Functions
(The commands are listed in the same order as the command descriptions that follow.)

Command	Function
PRINT	Initiate printout of specified data set on high-speed printer.
PUNCH	Initiate transfer of specified data set to punched cards.
WT	Initiate writing of specified data set on magnetic tape, with tape in format for offline printing.

PRINT Command

This command prints a data set on a high-speed line printer. The actual printing operation is not a part of the user's task but is performed nonconversationally by a bulk-output processor task.

Operation	Operand
PRINT	DSNAME=date set name [,STARTNO=first byte position] [,ENDNO=last byte position] [,PRTSP={EDIT | {1 | 2 | 3}[,HEADER=H] [LINES=lines per page] [,PAGE=P]}] [,ERASE=ERASE] [,ERROROPT={ACCEPT | SKIP | END}] [,FORM=paper form]

DSNAME

identifies the data set that is to be printed, and which must be defined within the current task by a DDEF command or must be cataloged.

Specified as: a fully qualified data set name.

STARTNO

specifies the byte position at which printing is to start for each data set record.

Specified as: one-to-six decimal digits.
System default: printing will start with the first byte of each record.

ENDNO

specifies the byte position at which printing is to stop for each data set record; this end byte will be printed.

Specified as: one-to-six decimal digits; must have a value greater than the STARTNO operand.
System default: printing will continue to the last byte of each logical record or until the printer line length is reached, whichever occurs first.

PRTSP

specifies the number of spaces to be skipped between lines.

Specified as: EDIT -- line spacing will be controlled by a character in the first byte position of each data set logical record. The control characters may be USASI or machine code (see Appendix H), but must be of the same type throughout the data set. The control character in each record is user-supplied.

1 -- one space between lines
2 -- two spaces between lines
3 -- three spaces between lines

Note: When EDIT is specified, the HEADER, LINES, and PAGE operands must not be specified.

System default: 1 is assumed.

HEADER

specifies that the first logical record of the data set is to be repeated on each print page as a header line. The first 132 bytes or the entire first record, whichever is smaller, will be used as the header.

Specified as: H
System default: no header will be printed.

LINES

indicates the number of lines to be printed on a page.

Specified as: one-to-four decimal digits; 9999 is the maximum
System default: 54 lines will be printed on each page.

PAGE

specifies that pages are to be numbered.

Specified as: P
System default: no page numbers will be assigned.

ERASE

specifies that the cataloged data set is to be erased from the catalog after the printing operation is finished.

Specified as: ERASE
System default: no erasure will be made.

ERROROPT

designates the action to be taken when an uncorrectable error is found while reading a data set record. This option applies only if the data set to be printed is on tape.

Specified as: ACCEPT -- error record will be accepted
SKIP -- error record will be skipped
END -- print operation will be terminated
System default: END is assumed.

FORM

designates the form number of the printer paper to be used.

Specified as: one-to-six alphameric characters.
System default: the installation's standard printer form will be used.

Functional Description: PRINT creates an independent nonconversational task, to which the system assigns a batch sequence number for possible reference by the user. The specified data set is printed as it appears. Invalid print characters will appear as blanks in the output. Data set records containing a read error (or an invalid control character, when the EDIT option is used) will be printed in hexadecimal on SYSOUT. When the data set resides on seven-track tape, the system makes the character adjustments required to ensure data validity.

If the user specifies a form number, the system will include that number in its instructions to the system operator when the printer is readied for operation.

The data set name specified may or may not be cataloged. If not, it is placed in the catalog until printing is completed and then erased, regardless of the ERASE option; if the data set name is cataloged, the ERASE option can be used to erase after printing is completed.

When EDIT is specified, the first byte of each logical record is assumed to be the byte following the control character, which is not printed and is not counted when determining where to begin printing a record.

If the data set to be printed was created via the DATA command, the first byte of each record contains an indicator of the origin of the record. PRINT translates the byte to a C if the record was entered through a card reader, and to a blank if it was entered through the keyboard. Unless the STARTNO operand is specified, this byte is printed as part of the record. If STARTNO is specified as 2, this byte is bypassed.

Cautions: If a data set that is affected by the task created by PRINT is used before the task is finished, the results will be unpredictable.

The PRINT command is valid for BSAM, VSAM, and VISAM data sets only. It cannot be used to print a member of a VPAM data set. However, a VPAM member can be copied with the CDS command and then the copy can be printed.

A BSAM data set must reside on magnetic tape; a VSAM or VISAM data set must not have undefined format records.

PRINT should not be used for an uncataloged data set that is awaiting bulk I/O, since PRINT automatically erases an uncataloged data set.

Programming Notes: The user may use the batch sequence number to identify his task when using the CANCEL command.

The user can also obtain a data set suitable for printing by using the WT command.

Example: The user, wants data set T44.REMOVE to be printed single spaced. The entire logical record is to be printed; no header or page numbers are wanted; 54 lines per page are wanted on standard printer forms; and the data set's catalog entry is not to be erased.

User: print t44.remove
System: (accepts task and assigns batch sequence number)

PUNCH Command

This command punches a previously created, public VAM data set into cards on a high-speed punch.

Operation	Operand
PUNCH	DSNAME=data set name [,CBIN=BINARY] [,STARTNO=first byte position] [,ENDNO=last byte position] [,STACK={1|2|3|EDIT}] [,ERASE=ERASE] [,FORM= paper form]

DSNAME

identifies the data set, which is to be punched and which must be defined within the current task by a DDEF command or must be cataloged.

Specified as: a fully qualified data set name.

CBIN

specifies punching in column binary format.

Specified as: BINARY
System default: punching will be in EBCDIC.

STARTNO

specifies the byte position at which punching is to start for each data set record.

Specified as: one-to-six decimal digits.
System default: punching will start with the first byte of each record.

ENDNO

specifies the byte position at which punching is to stop for each data set record; this end byte will be punched.

Specified as: one-to-six decimal digits, and must be a value greater than the STARTNO operand.
System default: punching will continue to byte 80 (or, if in binary, to byte 160) or end of record, whichever occurs first.

STACK

specified the stacker select or edit option:

Specified as: 1 - pocket number P1
2 - pocket number P2
3 - pocket number P3

EDIT - the first byte of each data set logical record contains a control character for stacker selection. This control character may be either USASI or machine code (see Appendix H), but must be of the same type throughout the data set. The control character in each record is user-supplied.

ERASE

specifies that the cataloged data set is to be erased from the catalog after the punch operation is finished.

Specified as: ERASE
System default: no erasure will be made.

FORM

designates the card form number of the cards to be used for punching.

Specified as: one-to-six alphameric characters.
System default: installation's standard card form will be used.

Functional description: This command results in the creation of an independent nonconversational task, to which the system assigns a batch sequence number for possible reference by the user.

The specified data set is punched as it stands, with no code conversions. The STARTNO and ENDO options allow selection of any contiguous field of up to 80 bytes of EBCDIC (or 160 bytes of binary) data from each record of the data set.

Input records containing an invalid control character, when the EDIT option is used, will be printed in hexadecimal form on system output (SYSOUT).

If the user specifies a form number, the system will include that number in its instructions to the system operator when the card punch is readied for operation.

The data set name may or may not be in the catalog. If not, it is placed in the catalog until punching is completed and it is then erased, regardless of the ERASE option. If the data set name is already in the catalog, the ERASE option can be used to erase the data set after punching is completed.

When EDIT is specified, the first byte of each logical record is assumed to be the byte following the control character, which is not punched and is not counted when determining where to begin punching the record.

If the data set to be printed was created via the DATA command, the first byte of each record contains an indicator of the origin of the record. PUNCH translates the byte to a C if the record was entered through a card reader, and to a blank if it was entered through the keyboard. Unless the STARTNO operand is specified, this byte is printed as part of the record. If STARTNO is specified as 2, this byte is bypassed.

Cautions: If a data set, affected by the task created by PUNCH, is used before that task is finished, the results will be unpredictable.

The PUNCH command is valid for VSAM and VISAM data sets only. It cannot be used to punch a member of a VPAM data set. (In the latter case, the member can be copied via the CDS command and the copy can then be punched.)

Programming Notes: The user may use the batch sequence number to identify his task when entering CANCEL command.

When the PUNCH command is used to punch a line data set, and the punched deck will subsequently be used as card-reader input to the RC command (issued by the operator) to recreate that line data set, the line numbers in the original data set should not be punched out. The RC command always prefixes a line number to each record of a line data set (see Appendix B).

Example: The user wants to punch characters 24 through 56 of each EBCDIC record in a data set GHOOTS9 and selects pocket 2. After completion of punching, the data set is to be saved. The usual card form is to be used.

User: punch ghoots9,,24,56,2
System: (accepts task and assigns batch sequence number)

WT (WRITE TAPE) Command

This command writes a data set on tape for eventual printing on a high-speed printer.

Operation	Operand
WT	DSNAME=current data set name, DSNAME2=tape data set name [,VOLUME=tape volume number] [,FACTOR=blocking factor] [,STARTNO=first byte position] [,ENDNO=last byte position] [,PRTSP={EDIT / 1 / 2 / 3} [,HEADER=H] [,LINES=lines per page] [,PAGE=P]] [,ERASE=ERASE]

DSNAME

identifies the data set that is to be written on tape in print format, and which must already be defined within the current task by a DDEF command or must be cataloged.

Specified as: a fully qualified data set name.

DSNAME2

specifies the data set name under which the data set is to be cataloged while it resides on the output tape.

Specified as: a fully qualified data set name.

VOLUME

specifies the volume identification number of the output tape.

Specified as: one-to-six alphameric characters.
System default: scratch tape will be used.

FACTOR

designates the blocking factor for records of the output tape.

Specified as: one-to-three decimal digits; the maximum blocking factor is 246.
System default: 30 is assumed

STARTNO

specifies, for each record, the byte position at which writing onto tape is to start.

Specified as: on-to-six decimal digits.
System default: writing will start with the first byte of each record.

ENDNO

specifies, for each record, the byte position at which writing onto tape is to stop. This end byte will be written.

Specified as: one-to-six decimal digits; value must be greater than the startno operand.
System default: writing will continue to the last byte of each logical record or until the printer line length (132 characters) is reached, whichever occurs first.

PRTSP

designates the number of spaces to be skipped between lines.

Specified as: EDIT -- line spacing will be controlled by a character in the first byte position of each data set logical record. The control characters may be USASI or machine code (see Appendix H), but must be of the same type throughout the data set. The control character in each record is user-supplied.

1 -- one space between lines
2 -- two spaces between lines
3 -- three spaces between lines

Note: When EDIT is specified, the HEADER, LINES, and PAGE operands must not be specified.

HEADER

specifies that the first logical record of the data set is to be repeated on each print page as a header line. The first 132 bytes or the entire first record, whichever is smaller, will be used as the header.

Specified as: H
System default: no header will be printed.

LINES

designates the number of lines to be printed on a page.

Specified as: one-to-four decimal digits (maximum 9999).
System default: 54 lines will be printed on each page.

PAGE

specifies that pages are to be numbered.

Specified as: P
System default: no pages will be numbered.

ERASE

specifies that the cataloged data set is to be erased from the catalog after the tape operation is finished.

Specified as: ERASE
System default: no erasure will be made.

Functional Description: WT results in the creation of an independent nonconversational task, to which the system assigns a batch sequence number for possible reference by the user.

The WT command processes input data sets that were created by using either VSAM or VISAM access methods. The tape data set, created by using the BSAM access method, is written in odd-parity with standard TSS/360 labels.

The selected field in each input data record is written on tape as a logical record or print line, in proper format for high-speed printing. Records will be blocked, if requested. The maximum blocked record length is 32,767 bytes. Input records containing a read-error (or an invalid control character when the EDIT option is used), are printed on SYSOUT, in hexadecimal form.

The input data set may or may not be cataloged. If not, it is placed in the catalog until writing is completed; then it will be erased, regardless of the ERASE option. If the data set was cataloged, the ERASE option can be used to erase the data set after writing is completed.

When EDIT is specified, the first byte in each logical record is assumed to be the byte following the control character, which is not printed or counted when the system determines where to begin printing a record.

If the data set to be printed was created via the DATA command, the first byte of each record contains an indicator of the origin of the record. PRINT translates the byte to a C if the record was entered through a card reader, and to a blank if it was entered through the keyboard. Unless the STARTNO operand is specified, this byte is printed as part of the record. If startno is specified as 2, this byte is bypassed.

Cautions: If a data set, affected by the task created by WT, is used before that task is finished, the results will be unpredictable.

WT is valid for VSAM and VISAM data sets only. It cannot be used for a member of a VPAM data set. However, a VPAM member can be copied via the CDS command and then the copy can be written onto tape.

Programming Notes: The user can use the batch sequence number to identify his task when entering the CANCEL command.

The user can also have a data set printed on-line by using the PRINT command.

Example: The user wants to create a tape, for off-line printing, that is double-spaced and uses the first record as a header. Bytes 20 to 130 of each record of data set RT.WINDER are to be printed on scratch tape. Pages are to be numbered and contain 60 lines, and the input data set is to be erased after it is written on tape. The tape data set is not to be cataloged and will be system blocked.

```
User:     wt rt.winder,,,,20,130,2,h,60,p,erase
System    (accepts task and assigns batch sequence number)
```

The user can manage his programs with these facilities, available through commands:

- Language processing
- Program control

The language processing commands enable the user to enter his source language data sets and have them processed into object modules. He can change and correct source language statements during processing. The language processing commands are described in Section 1.

The program control commands enable the user to call and initiate execution of object modules stored within the system, alter the paths of their execution, check their progress at any state of execution, modify them during execution, and pinpoint errors. These commands are described in Section 2.

SECTION 1: LANGUAGE PROCESSING

The user initiates source language processing by issuing one of three commands, listed in Table 10, for the desired language type.

Table 10. Language Processing Commands
(The commands are listed in the same order as the command descriptions that follow.)

Command	Language Type
ASM	assembler language
FTN	FORTRAN compiler language
LNK	linkage editor

ASM Command

This command invokes the assembler to assemble a source program module.

Operation	Operand
ASM	NAME=module name [,STORED={Y / N [,LINCR=(first line number,increment)]}] [,MACROLIB=({data definition name of symbolic portion, data definition name of index portion}[,...])] [,VERID=version identification][,ISD={Y|N}] [,SYMLIST={Y|N}][,ASMLIST={Y|N}][,CRLIST={Y|N}] [,STEDIT={Y|N}][,lSDLIST={Y|N}][,PMDLIST={Y|N}]

NAME
identifies the object module to be created. If the source program module (i.e., source language data set) is prestored, the user must have named it SOURCE.name
If it is not prestored, the system will automatically prefix SOURCE. to the source program module name. The listing data set will automatically be named LIST.name(0)

Specified as: the part of the source program module name that follows SOURCE., if the source program is prestored; otherwise, any one to eight alphameric characters, the first of which must be alphabetic. The object module name must be unique to the library in which it is stored; i.e., the module name must not be identical to other external entry points in that library.

STORED
specifies whether or not the source program module is prestored.

Specified as: Y - source program is prestored.
N - source program is not prestored.

System default: N is assumed.
When N is specified or STORED is defaulted, the user can specify the LINCR operand.

LINCR
specifies the line number to be assigned to the first line of the source language data set and the increment to be applied to succeeding line numbers.

Specified as: two three- to seven-decimal-digit numbers, separated by a comma and enclosed in parentheses; the last two digits in each number must be 0s.

System default: (100,100) is assumed.

MACROLIB
specifies the data definition name of the symbolic portion of the supplementary macro library to be used, and the data definition name of the index portion of that library. Both names must have been defined by a DDEF command within the current task. The user can specify a maximum of six libraries (i.e., six pairs of data definition names) which are searched in the opposite order in which they were specified; the system macro library is automatically made available to the user and is searched last.

Specified as: the data definition names defined in the previous DDEF commands.

System default: only the system macro library is used.

VERID
specifies the version identification to be assigned to the object program.

Specified as: one- to eight-alphameric characters, the first of which is alphabetic.

System default: the listing and the object modules are time stamped.

ISD
specifies whether an internal symbol dictionary (ISD) is to be produced.

Specified as: Y - ISD is produced.
N - no ISD is produced.

System default: Y is assumed.

SYMLIST
specifies whether a symbolic source program listing is to be produced.

Specified as: Y - listing is produced.
N - listing is not produced.

System default: N is assumed.

ASMLIST
specifies whether an object program listing is to be produced.

Specified as: Y - listing is produced.
N - listing is not produced.

System default: Y is assumed.

CRLIST
specifies whether a cross-reference listing is to be produced

Specified as: Y - cross-reference listing is produced.
N - cross-reference listing is not produced.

System default: N is assumed.

STEDIT
specifies whether the edited symbol table is to be listed.

Specified as: Y - edited symbol table is listed.
N - edited symbol table is not listed.

System default: N is assumed.

ISDLIST
specifies whether an ISD listing is to be produced.

Specified as: Y - ISD listing is produced.
N - ISD listing is not produced.

System default: N is assumed.

PMDLIST
specifies whether a program module dictionary (PMD) listing is to be produced.

Specified as: Y - PMD listing is produced.
N - PMD listing is not produced.

System default: N is assumed.

Functional Description: See "General Notes for Language Processing Commands," later in this section.

Examples:

1. The user wants a prestored source program (named SOURCE.TIP9) assembled, and he wants the ISD and a source program listing. The system macro instruction library is to be used:

User: asm tip9,y,,,y
System: (acknowledges receipt of the command; asks if modifications will be made)

User: n
System: (initiates language processing)

2. The user wants to process a prestored source program (named SOURCE.SRC) that is written in assembly language. He wants to make corrections, where required, and wants only a source program listing. The system macro instruction library is used:

```
User:       asm src,y,symlist=y,asmlist=n,isd=n
System:     (asks if modifications will be made)

User:       y
System:     (types # to invite the user's corrections)

Sys,User:   # 900,namea 1 14,asymb
            # 820, bc 15, namea
            #

User:       (presses RETURN key to end corrections)
System:     (inquires if there are further modifications)

User:       y
Sys,User:   # 800, be namea
            # 1025, dc a (namea)
            # 1025, anamea dc a(namea) (corrects above insertion)
            # (user presses RETURN key)
System:     (requests further modifications)

User:       n
System:     (inquires whether to continue processing)

User:       y
System:     (informs user of processing results)
```

3. The user wants to create his own macro instruction library for use with the assembler. The data definition name of the macro instruction library VISAM data set must be SOURCE; the data definition name of the macro instruction library index VSAM data set must be INDEX. The index is created to facilitate reference to the library by use of an IBM-utility program, SYSINDEX.

```
User:       ddef source,vi,mylib
            data mylib,i                        (User creates a VISAM
                                                data set; will use ) as
                                                header flag character)

Sys,User:   100 )macro1                         (line 100 is header of
                                                macro1; lines 200
                                                through 600, which are
                                                the text of macro1, are
                                                not shown)

            700 )macro2                         (line 700 is header of
                                                macro2; lines 800
                                                through 1500, which are
                                                text of macro2, are not
                                                shown)

            1600 %e

User:       ddef index,vs,myndx                 (User defines macro
                                                instruction library
            call sysindex                       index and calls SYSINDEX)

System:     (asks user to submit control
             statements)

User:       header=),length=8
            asm myprog, macrolib=-              (User assembles his
            (source, index)                     program)
```

FTN Command

This command invokes the FORTRAN compiler and compiles a source program module.

Operation	Operand
FTN	NAME=module name [,STORED={Y / N [,LINCR=(first line number, increment)]}] [,VERID=version identification] [,ISD={Y|N}] [,SLIST={Y|N}] [,OBLIST={Y|N}] [,CRLIST={Y N}] [,STEDIT={Y|N}] [,MMAP={Y|N}] [BCD={Y|N}] [,PUBLIC={Y|N}]

NAME
identifies the object module to be created. If the source program module (i.e., source language data set) is prestored, the user must have named it SOURCE.name
If it is not prestored, the system will automatically prefix SOURCE. to the source program module name. The listing data set will automatically be named LIST.name(0)

Specified as: the part of the source program module name that follows SOURCE., if the source program is prestored; otherwise, any one to eight alphameric characters, the first of which must be alphabetic. The object module name must be unique to the library in which it is stored; i.e., the module name must not be identical to other external entry points in that library.

STORED
specifies whether or not the source program module is prestored.

Specified as: Y - source program is prestored.
N - source program is not prestored.

System default: N is assumed.
When N is specified or STORED is defaulted, the user can specify the LINCR operand.

LINCR
specifies the line number to be assigned to the first line of the source language data set and the increment to be applied to succeeding line numbers.

Specified as: two three- to seven-decimal-digit numbers separated by a comma and enclosed in parentheses; the last two digits in each number must be 0s.

System default: (100,100) is assumed.

VERID
specifies the version identification to be assigned to the object program module.

Specified as: one-to-eight alphameric characters, the first of which is alphabetic

System default: the listing and the object modules are time-stamped.

ISD
specifies whether an internal symbol dictionary (ISD) is to be produced.

Specified as: Y - ISD is produced.
N - ISD is not produced.

System default: Y is assumed.

SLIST
specifies whether a source program listing is to be produced.

Specified as: Y - source program listing is produced.
N - source program listing is not produced.

System default: Y is assumed.

OBLIST
specifies whether an object program listing is to be produced.

Specified as: Y - object program listing is produced.
N - object program listing is not produced.

System default: N is assumed.

CRLIST
specifies whether a cross-reference listing is to be produced.

Specified as: Y - cross-reference listing is produced.
N - cross-reference listing is not produced.

System default: N is assumed.

STEDIT
specifies whether the edited symbol table is to be listed.

Specified as: Y - edited symbol table is produced.
N - edited symbol table is not produced.

System default: N is assumed.

MMAP
specifies whether a memory map is to be produced.

Specified as: Y - memory map is produced.
N - memory map is not produced.

System default: N is assumed.

BCD
specifies whether input contains the BCD (binary coded decimal) form of special characters.

Specified as: Y - input contains BCD form of special characters.
N - input does not contain BCD form of special characters.

System default: N is assumed.

PUBLIC
specifies whether the object module created has a public (rather than private) CSECT attribute.

Specified as: Y - module has public CSECT attribute.
N - module does not have public CSECT attribute.

System default: N is assumed.

Functional Description: See "General Notes for Language Processing Commands," later in this section.

Example:

The user wants to enter FORTRAN source language statements from his terminal. The object module is to be named RAADER; the starting line number and the increment are 100. Source program, object program, and cross-reference listings are requested.

```
User:      ftn raader,oblist=y,crlist=y,isd=n
System:    (requests each line by typing line number at the terminal,
           starting with 100)

Sys,User:  100 (10,5)a
           200 5 format (F6,2)
           300 b=atl
           400 write (20,5)b
           500 stop
           600 end

System:    (inquires if there are further modifications)

User:      n
System:    (inquires if processing is to continue)

User:      y
System:    (continues compilation)
```

LNK Command

This command invokes the linkage editor to link edit one or more object modules.

Operation	Operand
LNK	NAME=module name [,STORED={Y | N[,LINCR=(first line number, increment)]}] [,LIB=data definition name of library][,VERID=version identification][,ISD={Y|N}][,PMDLIST={Y|N}]

NAME

identifies the object module to be created. If the source program, consisting of the control statements that direct the linkage editor, is prestored, the user must have named it SOURCE.name
If it is not prestored, the system will automatically prefix SOURCE. to the source program module name. The listing data set will automatically be named LIST.name(0)

Specified as: the part of the source program module name that follows SOURCE., if the source program is prestored; otherwise, any one to eight alphameric characters, the first of which must be alphabetic. The object module name must be unique to the library in which it is stored; i.e., the module name must not be identical to other external entry points in that library.

STORED

specifies whether or not the source program is prestored.

Specified as: Y - source program is prestored.
N - source program is not prestored.

System default: N is assumed.

When N is specified or STORED is defaulted, the user can specify the LINCR operand.

LINCR
specifies the line number to be assigned to the first line of the source language data set and the increment to be applied to succeeding line numbers.

Specified as: two three- to seven-decimal-digit numbers separated by a comma and enclosed in parentheses. The last two digits in each number must be 0s.

System default: (100,100) is assumed.

LIB
identifies the library in which the new object module is to be included.

Specified as: the data definition name of the library.

System default: The last-mentioned library is assumed (i.e., the user library or a job library).

VERID
specifies the version identification to be assigned to the object program.

Specified as: one-to-eight alphameric characters, the first of which is alphabetic.

System default: the listing and the created modules are time-stamped.

ISD
specifies whether an internal symbol dictionary (ISD) is to be produced.

Specified as: Y - ISD is produced.
N - ISD is not produced.

System default: Y is assumed.

PMDLIST

specifies whether a program module dictionary (PMD) listing is to be produced.

Specified as: Y - PMD listing is produced.
N - PMD listing is not produced.

System default: N is assumed.

Functional Description: See "General Notes for Language Processing Commands," below.

Examples:

1. lnk abcd,n

The user wants to link edit modules into an object module named ABCD. Conversationally, he will enter all LNK operands and linkage editor control statements from the terminal. The linkage editor takes default values for the last five operands, which designate a starting line number and increment of 100, the module to be placed in the library currently at the top of the user's program library list, the listing to be time-stamped, an ISD, and no PMD listing.

2. lnk abcd,y,lincr=(1000,500),wxyz,verid=qxr,y

   The task described in example 1 is run conversationally with a prestored set of linkage editor control statements. He wants the object module placed in his own library named WXYZ.

General Notes for Language Processing Commands: The operands to be entered for each of the language processing commands are dependent upon whether the source program module (i.e., the data set containing source language statements) is prestored, and on the options selected by the user.

To be acceptable for language processing, a prestored source program must have line organization and must have been named SOURCE.name Source programs are automatically cataloged and retained by the system.

When source statements are submitted conversationally, or when they form part of the prestored SYSIN of a task, a source program will be constructed with line organization. Each physical line input to the system, either as a single card or as a single record of the line data set, becomes a physical record of the line data set (input length is limited to 120 characters). Continuation conventions for combining two or more physical records into a single logical statement for a language processor are specified by that processor.

From the user's standpoint, source language processing proceeds in one of four ways:

1. The task is nonconversational and the source program is prestored. The language processor picks up the source statements, line by line, and processes them. No corrections are made; any diagnostic messages are written for later reference by the user.

2. The task is nonconversational and the source program and the commands governing language processing appear, line by line, on SYSIN. In this case, a new source program is created as lines are read from SYSIN. A line number is prefixed to each line to serve as the key by which the line can later be identified. Any diagnostic messages are written for later reference by the user. The new program can be modified later.

3. The task is conversational, with a prestored source program. Successive lines from the source program are read and processed by the language processor. After all diagnostic messages for a single statement are available, they are written at the terminal and the user is invited to enter corrections. To indicate to the user that he can enter corrections, the system types a number sign (#) at the beginning of a new line and the keyboard is unlocked. The user may then enter a correction line, the first part of which is the line number that identifies the line being corrected, followed by a comma and the contents of the line:

   ```
   #500, DC A(EXAMPLE)
   ```

   This correction line is stored in the program, either as an insertion line or as a replacement line, and the system requests the next correction line by issuing #. To delete one or more lines, the user types, following #:

   ```
   D, line number
      or
   D, first line number, last line number
   ```

Such corrections change the source program permanently. To end corrections, the user presses the RETURN key in response to #. The correction lines are then processed by the language processor, and if no corrections are required for these, the next line is taken from the source program for processing.

4. The task is conversational and the user enters his source statements from the terminal (i.e., the source program is not prestored). The language processor, when ready for a source language line, writes a line number at the terminal, inviting the user to enter a line. The line the user types is stored in the source program being created, and is also passed to the language processor. The user can modify previously entered statements by typing after the system-issued line number:

   % line number

   and then continuing with the contents of his insertion or replacement line. He can delete a line or lines by typing after the system-issued line number:

   % D, line number
   % D, first line number, last line number

   The % identifies the line as a correction or deletion. When the user enters the next normal line (i.e., not prefixed by #), the previously collected modifications are sent to the language processor, and the normal line is stored in the source program. This line will then be picked up when the language processor has finished working on the modifications.

   If the user modifies a statement that has already been handled by the language processor, compilation will restart automatically. For a more detailed description refer to Assembler Programmer's Guide, FORTRAN Programmer's Guide and Linkage Editor.

   When the language processor issues a diagnostic message, the conversational user will be prompted with # to enter corrections. He can then enter insertions, replacements, and deletions as described for a conversational task with a prestored source program. He will continue to be prompted for corrections until he presses the RETURN key as the response to the # request; at that point he will be invited to enter his next source statement line.

The language processors indicate the number of a line that is in error, but do not issue the line itself. If the user wants to see the actual content of the line, he

1. Presses the ATTENTION key to interrupt source language processing,
2. Invokes the text editor and accesses the line in question,
3. Reissues the language processor command and resumes processing.

When the entire source program has been collected, the language processor finishes its analyses of source statements and may issue more diagnostic messages. In FORTRAN or assembler language and linkage editor processing, the processor asks the user (if he is in conversational mode) whether he wants to make modifications and restart, continue processing, or terminate processing.

When the user wants to continue, the next phase of the language processor is executed. The user is then informed, if no errors were found that prevented the processor from producing an object program module.

Finally, the object module is stored in the user's library (USERLIB), unless he has defined a job library. If the object module is to be stored in a library other than USERLIB, this library must be defined by

the user in his current task before he initiates source language processing. For the FORTRAN or assembler user, this library must be his most recently defined library. Supplementary macro instruction libraries, used during assembly, must also be defined before language processing is initiated. For additional information concerning definition of these libraries see the Assembler Programmers Guide or FORTRAN Programmers Guide.

If the linkage editor is the language processor, it places the object module in the library specified in its input operands. (Listing formats are shown in the Linkage Editor manual.)

The user has complete control of the listings that are printed. The system action for the listing data set varies, depending on whether or not the symbol given as the module name has been previously used or not. When this assembly, compilation, or link edit is the first one in which the symbol is used, the system establishes, in the user's catalog, a generation data group, called LIST.symbol, which will maintain two generations. The system also specifies that when the number of generations exceeds two, the oldest generation is to be erased. When the listing data set for the current run has been produced, the system catalogs it and makes it a new generation of the LIST.symbol generation data group.

When the symbol has been used previously as a module name, the system determines this and, knowing that a generation data group already exists, adds the listing as a new generation to the existing generation data group. Example: The third listing data set for a given symbol would become the latest generation (0); the second listing would become the (-1) generation; and the first listing would be erased.

Programming Notes: The user can change the number of generations maintained in the generation data group associated with a given symbol. Assume he has been working with a module called MYPROG, and that he currently has two generations in his LIST.MYPROG generation data group. He could then change the number of generations maintained in LIST.MYPROG.

1. Temporarily catalog the two generations as separate data sets (for this example MYPROG1 and MYPROG2).

   ```
   catalog list.myprog(0),u,,myprog1
   catalog list.myprog2(-1),u,,myprog2
   ```

2. Delete the system-defined generation data group, LIST.MYPROG.

   ```
   delete list.myprog
   ```

3. Define a new generation data group called LIST.MYPROG with, for example, five generations.

   ```
   catalog gdg=list.myprog,5,o,e
   ```

4. Add the two temporarily cataloged generations to the new LIST.MYPROG generation data group.

   ```
   catalog myprog2,u,,list.myprog(+1)
   catalog myprog1,u,,list.myprog(+1)
   ```

After the second CATALOG command is issued, MYPROG1 becomes the latest (0) generation and MYPROG2 becomes the (-1) generation; three more generations can be stored before MYPROG2 will be erased.

To obtain a printout of the desired listings after language processing, the user issues a PRINT command with a data set name

LIST.symbol(0) for the latest listing, or

LIST.symbol (-1) for the last previous listing, if two generations were specified.

The user can let the automatic erase logic associated with the generation data groups remove his unwanted listings. Or, he can issue the ERASE command or the ERASE option in the PRINT command to remove one or more generations.

SECTION 2: PROGRAM CONTROL

Program control commands provide the user with great flexibility for interacting directly with the execution of his programs. These commands, and the system functions they request, are shown in Table 11.

Table 11. Program Control Commands and Their Functions
(The commands are listed in the same order as the command descriptions that follow later in this section.)

Command	Function
LOAD	Place an object module in user's virtual storage without initiating execution.
UNLOAD	Remove specified object module from user's virtual storage.
CALL	Load and pass parameters to an object module and execute.
RUN	Initiate execution of loaded object module, resume execution of interrupted program; load and initiate execution of object module. (Restrictions on use of RUN will be given in its command description.)
GO	Resume execution of previously interrupted program.
REPEAT	After attention interruption, repeat last nonprompting message.
BRANCH	Dynamically change control path of program or resume execution at different location.
AT	Inform user when execution of program has reached designated instruction location; or make following statement dynamic.
REMOVE	Selectively delete previously entered dynamic statements (i.e., those that include AT).
IF	Make following statement conditional.
SET	Change contents of machine registers, values of program variables, virtual storage locations, or command symbols.
DISPLAY	Present values of variables, contents of machine registers and specified virtual storage locations to user's SYSOUT.
DUMP	Present values of variables, contents of machine registers and specified virtual storage locations to task's PCSOUT data set.
QUALIFY	Allow user to designate, before referring to group of internal symbols, program in which specified symbols are defined; thereafter, no need to explicitly qualify symbols.
STOP	Interrupt execution of user's program and display instruction location or FORTRAN statement number where interruption was handled.

Caution: Some of these commands are restrictive in the class of virtual storage they reference. The user may use all of these commands to reference his control sections that have been assigned to private read/write storage. However, a control section that has the read-only attribute, may be referenced in all the commands except SET. Public nonprivileged CSECTs may be displayed (via DISPLAY) or dumped (via DUMP), but the user cannot reference a public CSECT in a SET or AT command. A user may never symbolically access nonprivileged or privileged system CSECTs. Any violation of these restrictions will result in a diagnostic message and rejection of the command.

However, if a CSECT having a system or privileged attribute is loaded from the user library (USERLIB) or a job library (JOBLIB), all attributes are ignored. Private read/write storage will be assigned to the CSECT, and the system will not recognize any of the above restrictions.

The user can employ program control commands to:

- Explicitly and implicitly load and unload his programs;
- Initiate execution of his programs;
- Request output of the contents of data fields, instruction locations, and registers at any time during execution of his program;
- Modify instructions and variables within his program, at any stage of execution;
- Specify locations within his program where execution is to be stopped or started; when execution has been stopped, the user can issue additional commands before he resumes execution;
- Establish logical (i.e., true or false) conditions that allow or inhibit execution of other commands;
- Perform arithmetic computations.

Use of Command Statements

Program control commands are often conveniently expressed in command statements. For purposes of this discussion, three types of command statements will be considered: dynamic, immediate, and conditional statements.

Dynamic Statement: This is a command statement that contains an AT, which should appear first in the statement and should be the only one in the statement. Commands that precede AT in the statement are executed immediately; commands that follow AT are not executed until control arrives at the instruction location designated by the AT command. Only these commands can follow AT in a dynamic statement:

BRANCH	GO
CALL	IF
DISPLAY	SET
DUMP	STOP

If any other command appears in a dynamic statement, a diagnostic message is issued. Several dynamic statements can be effective at the same instruction location; the statements are processed in the order in which they were issued.

Immediate Statement: This is a command statement that does not contain an AT. Immediate statements are executed when they are entered. Any command, except AT, may appear in an immediate statement.

Conditional Statement: This is a command statement that contains an IF. Both immediate and dynamic statements can be conditional. The condition that IF specifies must be satisfied before the commands that follow are executed. Commands preceding the first IF command are executed without regard to IF. When more than one IF appears in a conditional statement, they will be evaluated as if they were joined by a logical AND. Any command may appear in a conditional statement.

Program Control Applications

To load an object module, the user can issue a LOAD or CALL command, or he can issue a direct call. The loading of one module may cause another module, which is implicitly referenced by the first module, to be loaded. Example: When a LOAD command is issued for module PGMA, which implicitly references module PGMB, PGMB is also loaded.

Following LOAD, the user may enter immediate command statements to alter the program before execution begins, or dynamic statements to alter the program during execution. The CALL command or direct call initiates execution for a loaded module, or loads and executes an unloaded module. To modify a program after it has been called (via CALL or direct call), the user presses the ATTENTION key and then enters his command statements. This procedure is not recommended for use of dynamic statements, since execution may have progressed past the point referenced by the AT command. He resumes execution with the GO or BRANCH commands.

When the user references an external symbol in a program that is not loaded, the program is loaded and the user can proceed as if he had entered the LOAD command. When the user references an external symbol that is not in any of the programs in the libraries available to him, the symbol is assumed to be a command symbol, which was defined via the SET command. A command symbol referenced in any other command is treated as a data reference to the last definition of the command symbol.

After execution has ended, the user can again issue command statements, or restart execution from a specified entry point, by using the CALL command.

The user can refer to internal program symbols in any loaded object module for which he requested an internal symbol dictionary (ISD) when that module was compiled or assembled; otherwise, he can reference only external symbols.

Dynamic statements remain enforced until a REMOVE command deletes them or until a program referenced by a dynamic statement is unloaded. A program is unloaded by an UNLOAD command, or when the only program that references it is unloaded. For example, if the loading of PGMA caused PGMB to be loaded, unloading PGMA would cause PGMB to be unloaded if PGMA was the only loaded module that referenced PGMB.

Note: If PGMA was referenced in any program control command, unloading PGMA would remove all dynamic statements that referred to it during the session. If PGMA is not referenced in a program control command, but PGMB is, all dynamic statements referencing PGMA are removed only if PGMA is also unloaded. A diagnostic message is issued when dynamic statements are removed because a module is unloaded.

Types of Address Specification

The user has broad addressing capabilities for referencing his programs by using variables and constants as operands for the program control commands.

VARIABLES: These are designated by their symbolic names, hexadecimal locations, or by register numbers.

1. Symbols: Program control commands use either external, internal, or command symbols.

   - External Symbols are defined within a program for reference at load or execution time. FORTRAN COMMON block names, function names, subroutine names, and the names of assembler language ENTRY and CSECT statements are external symbols. Example: An assembler program named PGM has these characteristics:

     2 control sections, named PGMCS and PGMPS

     2 ENTRY statements, named PGMEP and PGMEX. Then, these are valid external symbols:

     ```
     PGM
     PGMCS
     PGMPS
     PGMEP
     PGMEX
     ```

     Four external symbols are assigned to every FORTRAN object module:

     ```
     module name           (e.g., FTNPGM)
     CSECT name            (e.g., FTNPGM#C)
     PSECT name            (e.g., FTNPGM#P)
     module entry point    (e.g., FTNPGM#E)
     ```

     In program control commands, any of the external symbols may be referenced and, also, any function, subroutine, or COMMON block names. Variables referenced by external symbols have undefined type attributes.

   - Internal symbols are defined within a single assembly or compilation; FORTRAN statement numbers, FORTRAN data names, and symbols defined by the assembler statements are internal symbols. However, symbols defined by the assembler SET statement may not be used in program control commands.

     The user may refer to internal symbols only if he requested an internal symbol dictionary (ISD) when his program was assembled or compiled. Also, he must qualify each internal symbol to specify the program in which the symbol was defined.

     Note: When an ISD is requested for a FORTRAN compilation, optimum code is not generated.

     An internal symbol is qualified explicitly by preceding it with the name of the program in which it was defined and by following the program name with a period. When the defining program has not been processed by the linkage editor, only one level of qualification is required. Thus, for internal symbol IOSR defined in program PGM, the qualified symbol is:

     ```
     PGM.IOSR
     ```

     When the defining program has been processed by the linkage editor, two levels of qualification are required. The name of the program output by the linkage editor (first level) is followed by a period, the original name of the defining program (second level), and the internal symbol. Thus, for internal symbol IOSR, defined in program PGM, which has been processed by

the linkage editor into new program LEPGM, the qualified symbol is:

```
LEPGM.PGM.IOSR
```

An internal symbol may also be qualified implicitly, if its reference has been preceded by a QUALIFY command containing the necessary qualification. If internal symbol ABX has been defined in program PGMA, and a QUALIFY PGMA command has been entered, the internal symbol may be implicitly qualified only by ABX.

Note: If a program processed by the linkage editor contained an internal symbol which was identical to an external symbol in another program, explicit qualification is necessary to reference the internal symbol.

- Command Symbols are independent of the user's program and are defined by a SET command, which designates a symbol that the system cannot recognize as either an internal or external symbol. For example, in the command SET R = 5, if R is neither an external or internal symbol, the system designates R as a command symbol with a value of 5. The command symbol may now be referenced or modified by subsequent program control commands.

  When a command symbol has been defined, it is addressable for the user's entire terminal session; it will not be affected by unloading one of his programs. The command symbol may be retained for future terminal sessions by using the PROFILE command (see Part V, "User Profile Management").

  Note: If a program is loaded after a command symbol is defined, and the command symbol is identical to an internal or external symbol in the program, the command symbol is not recognized until that program is unloaded.

- %CSECT and %COM are two special symbols that may be used to refer to the unnamed assembler language control section and the FORTRAN blank COMMON, respectively. %CSECT may be used only as an internal symbol; %COM, as either an internal or external symbol.

- FORTRAN Statement Numbers are written by the user in the original source program, and should not be confused with the line numbers that are assigned to each source line by the compiler. Statements must be referred to by their numbers, not by line numbers. Executable statement numbers, used as internal symbols, can be incremented to refer to unnumbered statements. The increment must be an integer greater than 0, enclosed in parentheses, that immediately follows the statement number. The increment designated by (1) refers to the numbered statement itself. Therefore, 86(1) refers to numbered statement 86; 86(2) refers to the next executable statement following statement 86.

  Executable statements are arithmetic and logical assignment statements, control statements, and input/output statements. Nonexecutable statements are specification statements and subprogram statements; they should not be incremented.

  Examples of FORTRAN statements:

```
10      READ (1,20)A
20      FORMAT (F6.2)
        B = A*3.14
        WRITE (2,20)A,B
        GO TO 10
```

The third statement (B = A*3.14) is referenced by using 10(2). The FORMAT statement cannot be incremented since it is not executable.

Statement numbers refer to a statement's first line and any of its continuation lines; continuation lines should not be designated when using incremented statement numbers.

The integer 0 may be used to refer to a program's first executable statement when that is unnumbered. In the preceding example, if the READ statement were unnumbered, 0 could be used to refer to it; 0(2) would then refer to the second executable statement (B = A*3.14).

- Subscripted Symbols are internal symbols that refer to elements with an array. A subscript to an internal symbol must be one of these:

  - integer constants
  - integer variables
  - integer arithmetic expressions

  Symbols used in subscripts may also contain subscripts; and subscript symbols may also contain offsets. However, the data ultimately referenced by the subscript symbol must be an integer. Five levels of nesting (subscript and subscript, subscript and offset, offset and offset) are allowed.

  The subscript is enclosed in parentheses, following the internal symbol naming the array. One subscript may be used for each dimension of the array; multiple subscripts are separated by commas. A diagnostic message is issued if an evaluated subscript is not positive, an integer, 0, or is larger than the dimensions defined for the array.

  Examples:

  1. This two-dimension array contains three rows and five columns, and is defined by the internal symbol ARRAY.

     ```
      2   0   -7    5   13
     -2   1   15   -6    8
      0   1    3    9   -5
     ```

     ARRAY (2,4) refers to the array element at the intersection of row 2 and column 4.

     ```
     ARRAY (2,4)  =  -6
     ```

     ARRAY (4,4) would be invalid since it is outside the array.

  2. Consider this subscripted symbol:

     ```
     ARRAY (ARRAY (1,1),ARRAY (3,3))
     ```

     The subscript contains subscripted symbols that must be resolved first.

     ```
     ARRAY (1,1)  = 2
     ARRAY (3,3)  = 3
     ```

     When these values are substituted in the original expression,

     ```
     ARRAY (2,3)  =  15
     ```

3. Assume this table is defined by the symbol TABLE, and each item in the table contains a length attribute of 1.

```
TABLE      5
TABLE+1    3
TABLE+2    1
TABLE+3    4
TABLE+4    2
```

Now consider:

```
ARRAY(ARRAY(TABLE.(1),TABLE.(4)),ARRAY(TABLE.(2),TABLE.(3)))
```

This subscripted symbol has a subscript with an offset, nested within the subscript. Evaluation of the subscripted symbol starts inside the nesting and works outward.

```
TABLE.(1)=3
TABLE.(4)=2
TABLE.(2)=1
TABLE.(3)=4
```

Substituting:

```
ARRAY(ARRAY(3,2),ARRAY(1,4))
```

reduces the expression to one similar to Example 3. The final value is determined by continuing the process,

```
ARRAY (1,5) = 13
```

4. The subscripted symbol to be evaluated is

```
ARRAY (1+X/Z,X-Y*Y)
```

Assume that X=6, Y=2, Z=4.

The arithmetic expressions must be evaluated first.

```
1+X/Z = 1+6/4 = 1 + 1  =  2
```

```
X-Y*Y = 6-2*2 = 6 - 4  =  2
```

Therefore the expression reduces to

```
ARRAY (2,2)  =  1
```

Note: FORTRAN dimension variables and symbols, defined by assembler language DC or DS statements with duplication factors or multiple constants, are arrays. An array that is a dummy argument to a FORTRAN subprogram may be subscripted; the dimension of the array is defined in the subprogram. When an array has an adjustable dimension value, the value established at the latest execution of the subprogram is used. Assembler arrays are limited to a single dimension that is equal to the duplication factor multiplied by the number of multiple constants.

- Offsets reference a specific byte following a symbolic address. An offset of 1 references the next byte beyond the symbolic address. The number of bytes that constitute the offset is written after the symbol and its offset. The form is symbol, period, left parenthesis, offset, comma, number of bytes, right parenthesis:

```
SYMBOL.(OFFSET,LENGTH)
```

Length must be a positive integer. An offset may be one of the following:

- integer, hexadecimal or address constant
- integer or hexadecimal variable
- integer or hexadecimal arithmetic expression

The rules for nesting offsets are the same as for subscripts. However, a symbol cannot have both a subscript and an offset.

Thus

```
TAG.(ARRAY(2,3))
```

is a symbol with offset and is legal, but

```
TAG.(4) (ARRAY (2,3))
```

describes an invalid symbol that has both a subscript and an offset at the same nesting level.

Examples:

1. The 27th byte beyond DATA would be expressed as

   ```
   DATA.(27)
   ```

   or

   ```
   DATA.(X'1B')
   ```

   If four bytes are to be attributed to the data at the 27th byte from DATA

   ```
   DATA.(27,4) or DATA.(X'1B',4)
   ```

2. The user may readily reference data in a dummy control section (DSECT) by using the register offset. Assume general register 5 contains the address of the DSECT, and the field to be referenced has the symbol DATA associated with it in the DSECT; the location desired is

   ```
   DATA.(5R)
   ```

   Again, explicit length may be supplied

   ```
   DATA.(5R,8)
   ```

3. A four-byte field that is the 20th fullword field in a table whose address is A'DATA'.

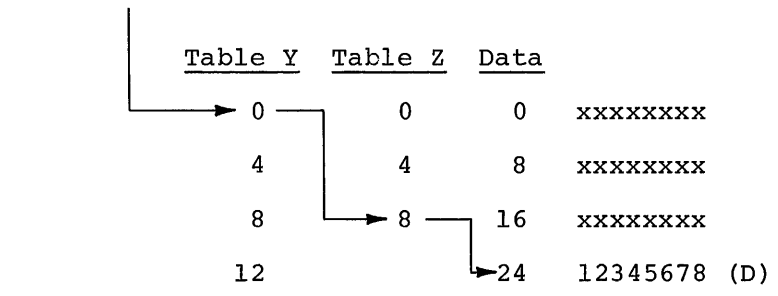
   ```
   .(A'DATA'+20*4,4)
   ```

   Note that the symbol to the left of the period is not required and is assumed to be location 0 if unspecified.

4. Assume the user's CSECT has address constant X that is the address of an entry in table Y, which contains a table-Z address that has the address of data D. The user can access D directly:

   ```
   .(.(.(.(X),4),4),8)
   ```

The sequence of events, from inside to outside of the expression, is shown in this schematic:



Thus

.(X) is the A'Y'

and the expression becomes

.(.(.(A'Y',4),4),8)

and .(A'Y',4) has a value equivalent to A'Z+8'

.(.(A'Z+8',4),8)

and at .(A'Z+8',4) assume a pointer to D, say A'D'

.(A'D',8)

and the final step yields the data

12345678

Note: When offsets are used with internal symbols, the referenced data has an undefined type attribute. A length attribute of one byte is assumed if an explicit length is not specified. The offset, when added to the location of an external symbol, must not reference a control section other than the one to which the symbol belongs.

5. It is possible to achieve a full virtual storage dump by specifying the range from location 0 to FFFFFF as offsets in the operand field of the DUMP command. Example:

```
dump .(x'0'):.(x'ffffff')
```

or

```
dump .(a'chbisa'):.(a'cevpas')
```

2. Hexadecimal Locations: These may be used in program control commands to refer to the contents of locations. The hexadecimal address of the location referred to is enclosed in apostrophes and preceded by L. The referenced virtual storage location must have been assigned to the user's storage.

Examples of hexadecimal addresses:

```
L'B000'
L'9FEC0'
L'9100'
L'8A36C214'
```

3. Registers: The user can use program control commands to refer to any of the general or floating-point registers. A reference to a general register is written as nR; n is a positive integer, less than 16, that identifies the register.

   A reference to a single-precision floating-point register is written as nE, n = 0, 2, 4, or 6. .A double-precision floating-point register is referenced by writing nD; n=0, 2, 4, or 6.

   Examples of register references:

```
3R  general-purpose register 3
2E  floating-point register 2, single-precision
6D  floating-point register 6, double-precision
```

4. Counter: A counter, associated with each dynamic statement, is incremented by 1 for each occurrence of the events specified in the statement. This counter must be referenced by the special character %. The value of the counter may be displayed or dumped; and can be used in forming expressions. The counter referred to must be the one associated with the statement in which it is referenced. Since % is not a user's variable, it cannot be changed by a SET command.

CONSTANTS: Five classes of constants are used in program control commands: integer, character, hexadecimal, floating-point, and address constants.

1. Integer Constant may be written as an optionally signed decimal integer. The length of an integer constant is not explicitly defined, but is determined from the expression in which the constant occurs. If the value of the number exceeds the permissible size, as determined by context,the number is truncated on the left.

   Examples of integer constants:

```
9327
-641
+1066
-67
```

2. Character Constant consists of letters, decimal digits, and special characters, enclosed in apostrophes. Also, any remaining unused combinations of the 256 valid card-punch combinations may be designated as a character constant. An apostrophe, used as a character in a character constant, must be represented by two apostrophes, even though only one will be put in storage. If the length of the constant is not appropriate, in the context used, the constant is truncated or filled with blanks on the right.

   Examples of character constants:

```
'$3.98'
'HOW ARE YOU?'
'I''M FINE'
```

3. Hexadecimal Constant is one or more hexadecimal digits (0 through 9, and A through F) preceded by an X and enclosed in apostrophes. A hexadecimal constant is either truncated or filled with 0s at the left, if its length is inappropriate for the context.

Examples of hexadecimal constants:

```
X'76543210'
X'FFFFFFFF'
X'ACE'
X'9FEC3'
```

4. Floating-Point Constant is a signed or unsigned decimal number in the principal part of the constant, which can be written with or without a decimal point. The decimal point can be at the beginning, end, or in any position within the decimal number, as appropriate.

   An exponent specifies a power of 10 by which the principal part will be multiplied during conversion. The decimal point may be omitted if an exponent is specified, in which case it is assumed to be located at the right-hand end of the decimal number. The exponent of a floating-point constant is either an E or a D, followed by a signed or unsigned decimal integer. An E indicates a single-precision floating-point number; a D indicates double-precision.

   The exponent may be omitted if the principal part contains a decimal point. When used, the exponent must follow the principal part of the constant. The magnitude of the exponent must be within the range of approximately 10^{-75} to 10^{75}. If the exponent exceeds the maximum, 10^{75} will be assumed; if it exceeds the minimum, 0 will be assumed.

   A floating-point constant will be converted to a normalized floating-point number. If the exponent of a floating-point number is omitted, the floating-point number is assumed to be single-precision.

   All of the following floating-point numbers are equivalent and will be converted to the same floating-point binary number:

   ```
   3.14159
   31.4159E-1
   314159.E-5
   314159E-5
   .314159E1
   ```

5. Address Constant consists of the character A followed by a symbol enclosed in apostrophes. The allowable symbols are: external symbol with or without offset, internal symbol with or without offset, and subscripted variable.

   The length of an address constant is always four bytes; its value is the address assigned to the symbol. Address constants are evaluated at the time they are used. The current value of any variable referenced in a subscript or offset is used in computing the value of the address constant. As a result, the value of an address constant that contains a subscripted or offset symbol may vary during program execution.

   Examples of address constants:

   ```
   A'PMG.TAG'
   A'NAME'
   A'ARRAY(I,J)'
   A'FTNPGM.100(36)'
   A'X.(4096)'
   ```

Operand Definitions

The terms used to describe the operands of program control commands are data location, data field, expressions (arithmetic and logical), instruction location, link-edited module name, object module name, and statement number.

1. Data Location may be specified as a symbol, a hexadecimal location, a register, or the special counter (%).

   Both fully and incompletely defined data locations may be referenced. Fully defined data locations have type and length attributes. Such locations include internal symbols without offsets, subscripted symbols, and floating-point registers.

   Incompletely defined data locations lack either the type or length attribute. A length attribute can be assigned to a symbol with offset; the offset following the period and left parenthesis is followed by a comma and a length specified as an integer or a hexadecimal constant that is greater than 0; then, the right parenthesis.

   Examples of symbols with offset and explicit length:

   ```
   Y.(X'EDC',4)
   Z.(12,8)
   A.(2,X'AF')
   ```

   Note that the offset may be defaulted and a length specified:

   ```
   Y.(,24)
   ```

   A length attribute may be assigned to a hexadecimal location by writing a colon that is followed by another, larger hexadecimal location. A diagnostic message is produced if any locations within the range have not been allocated to the user's virtual storage.

   Examples of hexadecimal data locations with explicit length attributes:

   ```
   L'9FEC0':L'9FEC7'
   L'9100':L'9103'
   ```

2. Data Field is a contiguous group of storage locations whose contents are to be displayed or dumped. These locations may be in the user's virtual storage or in registers. A data field may be a data location, an array, a control section, a symbolic range, a quoted string, or an arithmetic expression.

   An entire array is specified as a data field for display or dumping, if its name is written as an internal symbol without subscripting. Similarly, a control section name, written as an internal symbol without an offset, specifies the entire control section as a data field; so does a control section name written as an external symbol without an offset.

   A range of registers is specified as a data field by writing the numbers of the first and last registers to be displayed or dumped, separated by a colon, and followed by the character that identifies the register type:

   ```
   R = general registers
   E = floating-point registers with single-word form
   D = floating-point registers with double-word form
   ```

In specifying a range, general register 0 follows general register 15, just as floating-point register 0 follows floating-point register 6.

Examples of register-range specifications:

0:4R	general registers 0 - 4
14:3R	general registers 14 and 15, then registers 0 - 3
2:6E	floating-point registers 2, 4, and 6 in single-word format
6:2D	floating-point registers 6, 0, and 2 in double-word format

A data field may be specified by a symbolic range, which is written as two symbols separated by a colon. The storage location of the symbol to the right of the colon must be greater than the location of the symbol on the left; if not, a diagnostic message will be issued. Both symbols used to specify a data field must be either external symbols or internal symbols; one range may not be specified by an internal and an external symbol. When two internal symbols are used to specify a data field, both must have been defined within the same control section. External symbol ranges must be contained within user-assigned storage. Either or both of the symbols used to specify a data field may be offset, but may not have explicit lengths.

Examples of data fields specified by symbolic ranges:

```
A.BY:A.BX
PGM.LSF:PGM.LSA
LSF:LSA (if preceded by QUALIFY command)
ABX:ABX.(X'FFFF')
ABY:ABY.(256)
ABY:(24):ABY.(256)
```

3. Expressions used in program control commands are either arithmetic or logical; they are formed by using these operators:

Type	Operator	Meaning
Arithmetic	+	Addition
	-	Subtraction
	*	Multiplication
	/	Division
Logical	¬	Logical inversion or negation
	&	Logical intersection
	|	Logical union
Relational	>	Greater than
	<	Less than
	=	Equal to
	>=	Greater than or equal to

Type	Operator	Meaning
Relational (cont.)	< =	Less than or equal to
	¬ =	Not equal to
	¬ >	Not greater than
	¬ <	Not less than

- Arithmetic Expressions may be used as subscripts or offsets, as values to which variables are to be set, as values to be compared in relational expressions, or as values to be computed and displayed.

  The least complex arithmetic expression is a single constant or data location. However, an arithmetic expression may include any number of constants, data locations, and simpler arithmetic expressions that are related by arithmetic operators. The special character % may be used in an arithmetic expression to reference the dynamic statement counter.

  These rules must be followed in the formation of arithmetic expressions:

  1. Any arithmetic expression may be enclosed in parentheses.

  2. Arithmetic elements or expressions may be connected by arithmetic operators to form other arithmetic expressions, provided that no two arithmetic operators appear in sequence and no arithmetic operator is assumed to be present.

  3. An arithmetic element or expression preceded by a sign (+ or -) is permitted; the operators * and / must be preceded and followed by elements and/or expressions.

  4. All data locations connected by arithmetic operators must have 256 bytes or less and be aligned on the appropriate boundary.

  Arithmetic expressions that do not contain parenthesized terms are evaluated, left to right, in this order: (1) multiplication or division; (2) addition or subtraction. For example, the arithmetic expression

  ```
  PGM.A + PGM.B * PGM.C - PGM.D
  ```

  is evaluated as

  ```
  PGM.B * PGM.C      (denote result by X)
  PGM.A + X          (denote result by Y)
  Y - PGM.D
  ```

  Arithmetic expressions that contain parenthesized terms are evaluated by treating the innermost parenthesized term first. After all parenthesized terms have been evaluated, the remaining operations are performed as for nonparenthesized expressions. For example, the arithmetic expression

  ```
  PGM.A + (PGM.B - PGM.C) * PGM.D/PGM.E
  ```

is evaluated as

```
PGM.B - PGM.C       (denote result by X)
X * PGM.D           (denote result by Y)
Y/PGM.E             (denote result by Z)
PGM.A + Z
```

When division is performed in an integer arithmetic expression, the integer part of the quotient is retained and the fraction is discarded; thus, 13 / 2 = 6. The expression A*B/C may yield a different result than the expression

```
B/C*A
```

For example,

```
8 * 6 / 4 = 12
```

but

```
6 / 4 * 8 = 8
```

Examples of valid arithmetic expressions:

```
1.E-5
PGM.X.(4)
PGM.X/PGM.Y - 1
L'E000':L'E003' + 3
PGM.I * (PGM.J + PGM.K)
-Z.(,4)/%
```

The arithmetic method used to perform the operation is based on the type of the variables in the expression. Integer, floating-point, or logical arithmetic can be used in evaluation.

An undefined expression contains all undefined variables (e.g., external symbols and hexadecimal locations) or it contains two variables of different types.

If an undefined expression is used in a subscript, it is assumed to be integer. If an undefined expression has a variable that is longer than four bytes, the expression is assumed to be floating-point. The user is prompted in all other cases to provide the type of arithmetic to be performed.

An expression containing a constant can never be undefined. The data type of the constant is used to define the expression.

- Logical Expressions are used in a conditional statement, and take any of these forms:

  1. A single logical variable.

  2. Two or more logical variables connected by the logical operators & and/or | , denoting logical AND and logical OR, respectively.

  3. Two arithmetic expressions of the same type, connected by a relational operator.

A logical expression that contains a relational operator will have the logic value "true" if the condition expressed by the operator is met when the expression is evaluated. Otherwise, the expression will have the value "false."

The logical operator must be followed by a logical expression or term. Similarly, the operators & and | must be preceded and followed by logical expressions to form compound expressions.

Any logical expression may be enclosed in parentheses. Any compound logical expression to which the ¬ operator is to apply must be enclosed in parentheses.

Logical expressions that do not contain parenthesized terms are evaluated in this order:

1. Multiplication and division (* and /)
2. Addition and subtraction (+ and -)
3. Relational operations (> ,< , =, > =,< =, ¬=, ¬< , ¬ >)
4. Logical negation (¬)
5. Logical intersection (&)
6. Logical union (|)

When there is more than one operation of the same level, the operations are performed from left to right. For example, the expression

```
PGM.X/PGM.Y < 1.E-5 & PGM.Z  =  4
```

is evaluated as

```
PGM.X / PGM.Y       (denote result by A)
A < 1.E-5           (denote result by B)
PGM.Z = 4           (denote result by C)
B & C
```

This example would be evaluated as being "true" only if the data at PGM.X divided by the data at PGM.Y was less than 10^{-5} and the data at PGM.Z was the integer 4. The variables at PGM.X and PGM.Y must be floating-point data and the variable at PGM.Z must be integer data, to have the logical expression evaluated.

Parenthesized terms within logical expressions are evaluated in the same order. Then, when the expressions have been reduced (i.e., a single logical value has been assigned to each parenthesized term), evaluation is again performed in the order indicated. For example, the logical expression

```
(PGM.B = 2 & PGM.C = 3) | PGM.A = 1
```

is evaluated as

```
PGM.B = 2           (denote result by W)
PGM.C = 3           (denote result by X)
W & X               (denote result by Y)
PGM.A = 1           (denote result by Z)
Y|Z
```

In this example, the variable referenced must be integer data. The expression is "true" when the data at PGM.B = 2 and the data at PGM.C = 3, or when the data at PGM.A = 1.

Logical negation, indicated by the operator ¬, can be used preceding:

1. The relational operators =, >, <
2. A single logical variable, in which case the variable need not be enclosed in parentheses.
3. A compound logical expression, in which case the expression must be enclosed in parentheses.

Assuming both implicitly qualified symbols A and B are logical variables, and both C and D are arithmetic expressions, then the following are valid uses of the ¬ operator:

```
¬A
¬C = D & ¬ A
¬A | B
¬ (A | B)
```

The last two expressions are not equivalent. In the first case, the ¬ operator applies to the logical variable A; in the other case, the ¬ operator applies to the evaluated result A | B (i.e., if A is false and B is true, then A | B is true, and (A | B) is false).

4. Instruction Location refers to a statement within the user's source program. An instruction location is expressed either as the statement number of an executable FORTRAN statement or as an internal symbol in a source program written in assembler language. In either case, the user can apply an offset to the primary location designator. An explicit length will be ignored. When an internal symbol is used, it does not have to reference a location defined in the internal symbol dictionary (ISD) as an instruction or as a control section name.

   The user can express instruction locations as internal symbols within his program only if he requested an ISD when his program was last compiled or assembled. Otherwise, he must express them as external symbols (with or without offset) or as hexadecimal locations. In either case, the instructions must be on halfword boundaries.

   The ISD supplies the system with information concerning internal symbols. However, an ISD which is produced may not contain all of the information about the source program. For example, in assembler language usage, overlays caused by the ORG statement are not reflected in the ISD. If the user displays (via DISPLAY command) the storage locations affected by ORG statements, the contents will be correct but the assigned symbolic names will be misleading.

5. Link-Edited Module Name must precede the original program name, when qualifying internal symbols in a program that has been processed by the linkage editor.

6. Object Module Name is always the one assigned when the source module was compiled or assembled. When internal symbols are referenced, the object module name must always qualify the symbol. This name must be further qualified if the original program module was processed by the link editor.

7. Statement Number is assigned by the system to each statement containing an AT command. This number may be referenced in a REMOVE command.

Synonyms

Synonyms for program control command names and operands may be used. Examples of valid synonyms:

```
XYZ = LEPGM.PGM.IOSR
ABC = XYZ.(X'4C') (where XYZ is a synonym)
X = A + B * C
ABY = L'EF246'
ABX = ARRAY (I,J)
```

Whenever the system is processing an operand (such as a data location or a data field), and a synonym is recognized, the synonym is substituted. The operand derived by the substitution may also contain synonyms, which will be substituted one at a time. This procedure continues until all synonyms are resolved.

Synonym substitution occurs only for the first character string encountered when processing such operands as data location and data field. For example, for a data location defined by LEPGM.PGM.IOSR, synonyms would be substituted for LEPGM, but not PGM or IOSR.

Examples:

1. Assume the user has link-edited programs PGMA, PGMB, and PGMC that form a new program, LEPGM. Now the user wants to concurrently reference internal symbols within PGMA, PGMB, and PGMC with program control commands. Since only one qualification is allowed at one time, the user will be required to fully qualify all symbols in two of the three program modules involved.

   Suppose he now says:

   ```
   SYNONYM A = LEPGM.PGMB
   SYNONYM B = LEPGM.PGMC
   QUALIFY LEPGM.PGMA
   ```

   Now explicit qualification is much simplified; the user can reference symbols in PGMC merely by using B. as the qualifier. Thus the symbol X in PGMC can be referenced as

   ```
   SET B.X = X'00000000'
   ```

this is much simpler than

```
SET LEPGM.PGMC.X = X'00000000'
```

which would otherwise be required. Now, an expression such as

```
SET LEPGM.PGMA.Z = LEPGM.PGMB.Y + LEPGM.PGMC.X
```

can be stated as

```
SET Z +A.Y + B.X
```

2. The user has entered a QUALIFY command so that explicit qualification of external symbols is unnecessary. He then defines:

```
SYNONYM ARRAY=TABLE.
SYNONYM I=X'4C'
SYNONYM J=4
```

Then the expression:

```
DISPLAY ARRAY (I,J)
```

which would normally show an element of the array, is interpreted as

```
DISPLAY TABLE.(X'4C',4)
```

which will, instead, display an element of the table.

Note: Substitution is made for ARRAY, I, and J since each is a data location. Had ARRAY been explicitly qualified (PGM.ARRAY) then TABLE. would not have been substituted, since ARRAY was the second character string in the data location PGM. ARRAY.

LOAD Command

This command loads an object module, and all other object modules to which that module implicitly refers, into virtual storage, but does not initiate program execution.

Operation	Operand
LOAD	[NAME = entry point name]

NAME

identifies the module to be loaded.

Specified as: a module name or external entry point without offset.
System default: the last module referenced by the system is loaded.

Functional Description: When the LOAD command is executed, the system first searches the libraries on the task's current program library list to find the specified object module and allocates space for it in the user's virtual storage (i.e., loads the module). If that module is not implicitly linked to other modules, no further loading takes place. If that module is implicitly linked to one or more other modules, those modules and any other modules to which they are implicitly linked, are loaded by a similar search-and-allocate procedure. When a module to be loaded cannot be found, a diagnostic message is issued.

In the case of FORTRAN-written programs, a LOAD command specifying the main (or root) program causes the entire program to be loaded, because all FORTRAN subprogram modules are implicitly linked to the main module.

Assembler-written modules can be implicitly or explicitly linked to other modules. Explicitly linked object modules (e.g., explicitly called or loaded subroutines of a program's main module) are not loaded when a LOAD command is executed; they are loaded one at a time during execution as each explicit linkage is processed.

Caution: A FORTRAN COMMON BLOCK program must be loaded by module name, not COMMON BLOCK NAME, since only the module name can be found by the dynamic loader.

Programming Notes: The LOAD command can be used to load object modules so that dynamic statements can be inserted prior to execution.

Example: Load module ABC, and all modules to which it implicitly refers.

User: load abc
System: (loads ABC and all implicitly linked modules into virtual storage)

UNLOAD Command

This command removes a module, and all other modules to which it (and only it) implicitly or explicitly refers, from virtual storage.

Operation	Operand
UNLOAD	[NAME = entry point name]

NAME

identifies the module to be unloaded.

Specified as: a module name or external entry point without offset.
System default: the last module referenced by the system is unloaded (i.e., the current module).

Functional description: The UNLOAD command invokes the dynamic loader, specifying the explicit symbol that is specified in the NAME operand of the command. If NAME is not specified, the last module loaded or called by the command system is unloaded.

The specified object module is unloaded from virtual storage; any object modules that are referred to only by that specified module are also unloaded.

The specified module is not unloaded if other object modules are currently referring to it. The user is informed of this in a system message, so he can reissue the UNLOAD command later, if desired.

Programming Notes: Ordinarily, an object module that is called by a direct call is not automatically unloaded upon exit. The UNLOAD command can be used to remove these modules from virtual storage.

Example: Unload a module named ABC.

User: unload abc
System: (unloads ABC and all modules that only ABC implicitly refers)

CALL Command

This command invokes an object module (or a procedure).

Operation	Operand
CALL	[NAME=entry point name [,module parameters]]

NAME

identifies the module to be invoked.

Specified as: a module name or external entry point without offset.
System default: the last module referenced by the system is called.

module parameter

specifies the parameters associated with the module being called; when a module expects parameters, all parameters must be specified, including the commas representing null values, whether or not the parameters are normally defaultable.

Specified as: the parameters, separated by commas, expected by the module.
System default: the module called does not expect parameters.

Functional Description: CALL invokes the dynamic loader and passes to it the name of the module specified; if a module was not specified, CALL passes the module name most recently referenced by the system. Modules implicitly referenced by the specified module are also loaded; the called module is invoked via standard type-1 linkage. CALL passes control to a module that is already loaded. When the specified module cannot be found, a diagnostic message is issued.

Caution: If the module called during the execution of a dynamic statement has dynamic statements embedded in it, the results are unpredictable.

Programming Notes: A module can be invoked by either the CALL command or a direct call. A direct call follows the command system symbol-resolution process in which procedures take precedence over modules. If a module and a procedure have the same name, the procedure is invoked by a direct call. In this case, a CALL command must be used to invoke the module.

CALL may be used to initiate execution of a module that is already loaded.

Examples:

1. The user wants to compile and then execute program MYPRG.

   User: ftn myprg
   System: (compiles and stores myprg)

   User: call
   System: (invokes MYPRG)

2. The user wants to call module XYZ and pass five parameters.

   User: call xyz, parl,,,par4,
   System: (invokes XYZ and places a pointer to the parameter list in register 1)

3. The user wants to call module XYZ and pass one real-value parameter.

   User: call xyz '$$*#@%'
   System: (invokes XYZ and places a pointer to the parameter list in register 1).

Direct Call

When the user wants to load and immediately execute an object program (or procedure), he may do so by entering the module name and the operands expected as parameters by the module. The system then loads the module, also loading any implicitly referenced modules, and passes control to the explicitly loaded module.

When a procedure and a module have the same name, the procedure is called. The CALL command would invoke the module.

When the specified module or procedure cannot be found, a diagnostic message is issued.

When a module expects parameters, all parameters must be specified, including commas for null values, whether or not the parameters are normally defaultable.

Caution: A direct call is not permitted in a dynamic statement.

Examples:

1. Load and execute module ABC.

   User: abc
   System: (invokes ABC)

2. Load module ABC, pass parameters X, Y, and Z, and execute.

   User: abc x, y, z
   System: (invokes ABC and places a pointer to the parameter list in register 1)

RUN Command

This command initiates or resumes object program execution, or causes a standard call to another object program. RUN is interpreted by the system as a CALL or GO command. In every instance where RUN might be used, GO or CALL is recommended.

Operation	Operand
RUN	[LOC = entry point name]

LOC

specifies the location at which execution is to begin or resume.

Specified as: a module name or external entry point without offset.
System default: RUN performs as GO, and resumes execution from the point at which the program was last interrupted or stopped.

Functional Description: RUN is a procedure which breaks down into a GO or a CALL command. When RUN is specified with an operand, the system executes a CALL command; without an operand, the system executes a GO command.

Examples:

1. The user wants to load and execute his FORTRAN program named FTNPGM.

   User: run ftnpgm (CALL would be equivalent and is recommended)
   System: (verifies that the object module is running)

2. The user wants to load and execute an object program, but wants to enter some dynamic statements prior to initiating execution.

   User: load pgm
   System: (verifies loading of PGM)

   User: (enters dynamic statements followed by RUN pgm; CALL would be equivalent and is recommended)
   System: (verifies that PGM is running)

3. The user has interrupted his program and wants to resume execution.

   User: run (GO would be equivalent and is recommended)
   System: (resumes execution of interrupted program)

GO Command

This command resumes execution of a previously interrupted object program (or command).

Operation	Operand
GO	

Note: There are no operands.

Functional Description: GO gives control to the most recently interrupted object program or command. When GO is followed by other commands in a command statement, the succeeding commands are ignored after GO is executed.

Caution: GO in a dynamic statement is meaningless; a diagnostic message will be issued. In an immediate statement, GO must appear last since the commands following are ignored; no diagnostic message will be issued.

Programming Notes: GO is meaningful only when it follows an attention interruption; otherwise it is ignored.

Example: In executing his program ABC, the user wants to interrupt execution and modify his program.

User: call abc
System: (invokes ABC)

User: (presses ATTENTION key)
System: !

User: set 5r=6;go
System: (resumes executing ABC from point of interruption)

REPEAT Command

This command requests the system to display the last-issued nonprompting message following an attention interrupt.

Operation	Operand
REPEAT	

Note: There are no operands.

Functional Description: REPEAT displays the last nonprompting message issued to the user's terminal; the interrupted program is then resumed.

Programming Notes: REPEAT is useful when the user interrupts his task while a message is being output to his terminal. This command is meaningful only following an attention interruption; otherwise, it is ignored.

Example: The user has interrupted his conversational task while a message was being displayed, and he wants to see the entire message.

User: repeat
System: (displays last-issued nonprompting message and resumes execution of the interrupted program)

BRANCH Command

This command changes the control path of a program or resumes execution of a program at a different location.

Operation	Operand
BRANCH	INSTLOC=instruction location

INSTLOC

specifies the location of an instruction within an object module at which execution is to resume.

Specified as: an explicitly or implicitly qualified internal symbol, with or without offset; an external symbol, with or without offset; or a hexadecimal address.

Functional Description: BRANCH resumes execution, at the specified location, of a program that has been stopped; or initiates execution, as part of a dynamic statement, at the specified location.

Cautions: In a command statement containing more than one command, BRANCH should be the last command, since any subsequent commands will be ignored and no diagnostic will be issued.

BRANCH cannot be used as a means of invoking a program.

Programming Notes: When the user wants to use internal symbols in the INSTLOC operand, he must have requested an ISD when assembling or compiling his program.

Examples:

1. The user, having stopped execution of his program (which has an ISD), wants to resume execution at an instruction location labeled with the internal symbol LOCA.

   User: branch pgm.loca
   System: (resumes execution at LOCA)

2. The user wants to alter the execution path of his program (PROG) from location PTA to PTC.

   User: qualify x
   at a; branch c
   x
   System: (passes control to C when execution reaches A)

AT Command

This command requests notification when execution of an object program reaches specific instruction locations. AT also designates the object program instruction locations at which the commands following AT in the dynamic statement are to be executed.

Operation	Operand
AT	instruction location [,...]

Note: Keyword operand format is not valid.

instruction location

specifies the location of an instruction within an object module.

Specified as: an internal or external symbol with or without offset or subscript, or a hexadecimal address.

Functional Description: AT becomes effective when control arrives at the instruction location specified in the operand, but before the instruction at that location is executed. The system assigns, to each dynamic statement, a number, which may be referenced by REMOVE.

When an AT command becomes effective, a standard output, which includes the instruction location where the command became effective, program status information, and the statement number, is presented to the user. The program status information includes the virtual storage location of the instruction being executed, the instruction length code, the condition code, and the program mask. If the user refers to an instruction location in a shared program or a system program, a diagnostic message is issued and the command is ignored for that location. A diagnostic is also issued if the instruction location contains a supervisor call operation requiring parameters that must follow the SVC.

The counter, referred to by the special character %, is assigned to a dynamic statement and is incremented by 1, when the program arrives at an instruction location designated in the AT command. The counter is incremented even when the dynamic statement is conditional. The counter may be used as an operand in the other program control commands within the statement. The AT command alone will interrupt but not stop program execution.

Caution: The user should not designate an instruction location that was modified by program execution; if he does, the results are unpredictable.

Programming Notes: If AT specifies FORTRAN statement numbers as instruction locations, the numbers must designate executable FORTRAN statements and not format statements.

Example: The user wants to be informed when his program reaches specified locations.

```
PGM.S1
PGM.S3.(4)
FTNPGM.98
FTNPGM.98(5)
```

To accomplish this,

User: at pgm.sl,pgm.s3.(4),ftnpgm.98(5),ftnpgm.98
System: 00001

Assuming execution of the program is initiated, when control arrives at any of the instruction locations, the user is notified.

System: AT FTNPGM.98 PSW 1 3 0 0003F076 0001

where:

FTNPGM.98 = instruction location

PSW 1 3 0 003F076 = program status

00001 = statement number assigned by the system.

REMOVE Command

This command deletes previously issued AT commands or dynamic statements.

Operation	Operand
REMOVE	statement number [,...]

Note: Keyword operand format is not valid.

statement number

identifies an AT command or a dynamic statement that is to be deleted.

Specified as: the number assigned by the system when the AT command or dynamic statement was entered.

Functional Description: REMOVE permanently cancels all dynamic statements or AT commands whose numbers are specified as operands.

Caution: A REMOVE command may not appear in a dynamic statement.

Example: The user wants to remove dynamic statements 10, 2, and 4.

User: remove 10,2,4
System: (deletes dynamic statements)

IF Command

This command, included in a command statement, creates a condition that must be satisfied if the remaining commands in the statement are to be executed. IF can be combined with any other command or commands in a conditional statement to establish any valid condition.

Operation	Operand
IF	condition

Note: Keyword operand format is invalid.

condition

specifies a condition that must be true to allow execution of commands that follow the IF command in the conditional statement.

Specified as: a logical expression.

Functional Description: If the command statement containing the IF command also contains an AT command, the logical expression is evaluated only when the instruction locations specified in the AT command are reached. The counter associated with each dynamic statement containing an AT command, referred to by the special character %, is incremented by 1, when the specified instruction location is reached, whether or not the IF condition is true. When more than one IF command appears in the same conditional statement, the IF commands will be evaluated as if they were joined by a logical AND. Example:

```
if X< 0; display X; if Y< 0; display Y
```

(Y will only be displayed when both X and Y are less than 0.)

Programming Notes: An IF command may stand alone, but it performs no useful purpose. If the condition is true, there are no further actions to be performed. If the condition is false, the remainder of the statement would have been ignored. In either case, the results appear to be the same. The dynamic statement counter can be used in forming a logical expression for the IF command. The counter, referred to by %, may be used to control the frequency at which, or the interval through which, the statement containing IF is effective. In nondynamic statements the counter has a constant value of 1.

Examples:

1. The user wants to test a logical condition and, if that condition is true, to issue other program control commands. The condition is true only when the value of his internal symbol variable PGM.NUM is less than or equal to 14.

```
User:     if pgm.num ≤ 14; display pcm
System:   (evaluates the logical expression and executes DISPLAY
          only if the condition is true)
```

2. The user wants to execute more program control commands every fifth time.

```
User:     at p.x; if % = (%/5)*5;...
System:   (assigns a number to the dynamic statement)
```

SET Command

This command changes the contents of a data location.

Operation	Operand
SET	{data location = value} [,...]

Note: Keyword operand format is not valid.

data location

identifies a location whose value is to be changed.

Specified as: a symbol, hexadecimal location, register, or command symbol.

value

specifies the value to which the data location is to be set.

Specified as: an arithmetic expression, a character string, or the name of a data location.

Functional Description: The SET command changes the contents of each specified data location to the value specified on the right of the corresponding equal sign.

The expression is evaluated using integer, floating-point, or logical arithmetic. All constants in an expression must agree in type. All variables should agree in type but, if they do not, the type is assumed by the system to be an integer (1,2, or 4 bytes), floating-point (8 bytes), or hexadecimal (length defined implicitly or explicitly). After SET is executed, a data reference in a subsequent command results in obtaining the new value.

Cautions: Although the user may set one complex variable to the value of another complex variable, no arithmetic can be performed between two complex variables. This restriction also applies to variables in packed decimal number format.

The operand of the SET command may never reference read-only or privileged storage. Since the FORTRAN compiler automatically assigns the read-only attribute to the control section containing instructions and constants, the FORTRAN user can not reference the CSECT as the data location of the SET command. When the expression contains more than one operand, the lengths of the operands must be compatible (i.e., floating-point variables must be 4 or 8 bytes; integer and logical variables must be 1,2, or 4 bytes). The length of the result of the expression should agree with the length of the data location to the left of the equal sign.

The format =X'C1C2C3' is acceptable for hexadecimal representation; the format ='ABC' will give the same result. However =C'ABC' will result in a diagnostic message.

Examples:

1. The user wants to set two 4-byte variables with qualified internal symbols of I and K to the values of 33 and 176, respectively.

   User: set i = 33, k = 176
   System: (sets values)

2. The user wants to set a 6-byte field to read 'system'.

   User: set field = 'system'
   System: (sets value at FIELD to E2E8E2E3C5D4)

3. The user has two variables that he wants to add, placing the result in general register 8. Both variables were assigned hexadecimal types in the assembly program. Variable X was defined as two bytes in length; variable Y as eight bytes. The user wants to refer only to the first two bytes of Y.

   User: set 8r = x + y. (0,2)
   System: (sets value)

DISPLAY Command

This command prints the contents and names of specified data fields on SYSOUT.

Operation	Operand
DISPLAY	data field name [,...]

Note: Keyword operand format is not valid.

data field name

identifies one or more data fields to be displayed.

Specified as: the name of a data location, an array, a control section, or a symbolic range; an arithmetic expression or a quoted string.

Functional Description: The contents of each specified data field, identified by the name entered in the operand field, are printed. The format of this printout is established by the system, according to the type and length attributes of the data field. If the data field type is not defined, it is assumed to be hexadecimal. If the user's task is conversational, the data field is printed at the terminal; if the task is nonconversational, it is entered on SYSOUT. When a control section name, used as an internal symbol, is entered as an operand of DISPLAY, the entire control section is automatically formatted in accordance with information in the internal symbol dictionary and is printed in symbolic form in assembler language. When a control section name is used as an external symbol in DISPLAY, the entire control section is printed in hexadecimal.

When a symbolic range of internal symbols, without offsets, is entered as an operand of DISPLAY, the specified range is automatically formatted and printed in symbolic form. If either internal symbol in a symbolic range has an offset, the output is in hexadecimal.

Programming Notes: Arithmetic operations may be executed with the DISPLAY command if the user does not want to have the result saved in storage. The type and length of the operands must be compatible.

When the user is in conversational mode, he can terminate the printout by pressing the ATTENTION button at his terminal. If more than one data field had been specified in one DISPLAY command, the next data field is displayed; otherwise, control is passed to the terminal.

Examples :

1. The user wants to print a header and the contents of register 6E.

   User: display c'register 6', 6e
   System: REGISTER 6 .27182818E + 01

2. The user has a 5 x 5 integer array and wants to have the first ten elements displayed, as well as the elements referenced by the subscripts I and K.

```
User:     display pgm.array (1,1): pgm.array (5,2),pgm.array-
          (pgm.I,pgm.K)
System:   PGM.ARRAY (1,1): PGM.ARRAY (5,2) =
          (1,1)  4  5 -8 1  6
          (1,2)  9 -6  3 22 7
          PGM.ARRAY (4,5) = -16
```

Note: The elements may be referenced symbolically, but the system produces the actual subscript values.

DUMP Command

This command places the contents and names of specified data fields in the data set with a data definition name of PCSOUT.

Operation	Operand
DUMP	data field name [,...]

Note: Keyword operand format is not valid.

data field name

identifies one or more data fields to be placed in the PCSOUT data set.

Specified as: the name of a data location, array, control section, symbolic range; a quoted string or an arithmetic expression.

Functional Description: The contents of the specified data fields are output to the PCSOUT data set. The format of the results is the same as for DISPLAY.

Programming Notes: DUMP should be used for large amounts of data.

There is only one PCSOUT data set per task. It is organized as a line data set. The user has facilities for printout control of the data produced by the DUMP command. This procedure is recommended:

```
DDEF      PCSOUT,VI,DSNAME = name

DUMP      data field name

RELEASE   PCSOUT

PRINT     name,,,EDIT
```

The DDEF command (or macro instruction) must be used to define and catalog the PCSOUT data set (i.e., specify PCSOUT as the data definition name) before DUMP is issued. If no definition has been given, the user is prompted to issue one. Refer to Appendix F for a detailed description of the DDEF command.

The user can specify the ERASE option in the PRINT command to remove the PCSOUT data set from his catalog.

Examples:

1. The user wants to output the contents of an entire control section to his PCSOUT data set; assume this is the first use of DUMP in this task.

```
User:     ddef pcsout, vi,dsname = list.pcsout
          dump pgm.csectl
System:   (outputs data)
```

2. The user wants to see the contents of a control section, but does not have an ISD for the program module. He issues a DUMP, using the control section name as an external symbol; however, he fails to issue a LOAD command for the object module.

   User: dump csect
   System: (loads the module and outputs data)

QUALIFY Command

This command allows the user to reference the internal symbols within a loaded object module, without requiring use of the fully qualified name. Internal symbols may be designated as either implicit (qualifying prefix is omitted, but value is assumed) or as explicit (entire value, including prefix is named).

Operation	Operand
QUALIFY	MNAME=[link-edited module name.]object module name

MNAME

identifies an assembled or compiled program (object module); and, optionally, a module processed by the linkage editor.

Specified as: an object module name and, optionally, a link-edited module name; the module names must be separated by a period.

Functional Description: An ISD must have been requested when the original program was assembled, compiled, and (if applicable) processed by the linkage editor. QUALIFY enables the user to reference, implicitly, the program's qualified internal symbols in the ensuing commands or statements.

Caution: Only one QUALIFY command can be in effect at one time; each QUALIFY command overrides any previous ones.

Examples:

1. The user wants to reference, implicitly, the qualified internal symbols in the program named PGMF, which is a part of the link-edited program named PGML.

   User: qualify pgml. pgmf
   System: (accepts command)

   Note: The user may, thereafter, reference internal symbols in this program in implicitly qualified form.

2. The user wants to qualify his internal symbols in program PGMF, but he failed to request an ISD with his assembly or compilation.

   User: qualify pgmf
   System: (informs user that the module has no ISD; the user must use external symbols in his references to program PGMF)

3. The user wants to qualify his internal symbols defined in program PGMF, which is part of the link-edited program named PGML. However, in entering the QUALIFY command, he neglects to specify the name of the original assembly or compiler module.

User: qualify pgml
System: (informs user he needs two levels of qualification)

User: qualify pgml.pgmf
System: (accepts command)

STOP Command

This command stops execution of an object program and prints out the current instruction location and program status information.

Operation	Operand
STOP	

Note: There are no operands.

Functional Description: The STOP command causes the output of two units of information at the user's terminal.

1. Current location in the object program; i.e., the instruction location, expressed symbolically, at which execution is stopped.

2. Program status information (e.g., the condition code, program mask, and instruction length code).

If the internal symbol dictionary (ISD) is not available, the symbolic instruction location is expressed in terms of the control section name and a hexadecimal offset. If the ISD is available, the location is expressed in terms of an internal symbol plus hexadecimal offset. The nearest internal symbol, plus an offset (in bytes) is output for assembly language programs. For FORTRAN programs, it is the nearest statement number, with an increment to indicate which statement, following the numbered statement, has control.

Caution: STOP should appear last in a dynamic statement since any subsequent commands are ignored and no diagnostic is issued.

Programming Notes: After an object program has been halted, the user can cause resumption of execution with the GO command.

Examples:

1. The user wants to learn the status of his program when execution reaches a specified point.

   User: at ftnpgm.100(4); stop
   System: (when execution reaches FTNPGM.100(4), system replies by giving statement number assigned to above statement; also gives program status information)

2. The user interrupted his FORTRAN object program during execution, by pressing the ATTENTION key at his terminal. He wants to know which statement is currently being executed. The user requested an ISD as an option during compilation.

   User: stop
   System: STOP AT FTNPGM.100(4) PSW 1 1 0 0003E0F4

Program Control Examples

The internal symbols in all the following examples are implicitly qualified, since a QUALIFY command was entered with the name of the defining program.

1. The user wants to display the contents of all general registers and floating-point registers in doubleword format, when his program reaches the instruction location ERREXT. He also wants the contents of the virtual storage locations, in the range from TOP to BOT, to be put into his PCSOUT data set, when program control reaches the ERREXT location.

```
at errext; display 0:15r, 0:6D; dump top:bot
```

2. The user wants to change the value of variable POINT to the address of the external symbol DATA, when his program arrives at instruction location TAGA.

```
at taga; set point = a'data'
```

3. The user wants to display a table, TAB, every tenth time through the loop ENTAB. When the loop is executed 100 times, he wants to dump control section BLDTAB.

```
at entab; if % = (%/10) *10; display tab; -
if % = (%/100)*100; dump bldtab
```

4. The user wants to use program control commands to produce input and output to his program. He wants to make some computations, using the sequential numbers 50 to 500. At statement number 10 he sets up a constant, INPUT, using the variable A, which was previously initialized at 0. At the end of each computation, which is statement number 80, he wants to see the result, OUTPUT.

```
at 10; set input = a + 50; set a = a + 1; -
if input = 500; stop

at 80; display output; branch 10
```

5. The user has assembled his program and discovered that he has forgotten to provide a label (TAGA) for the instruction

```
L  2,XYZ
```

which is located at hexadecimal location 124 and referenced by

```
B  TAGA
```

which is at hexadecimal location 176. By using program control commands he can fix his program temporarily without reassembling.

```
at  csect. (x'124'); branch  csect.(x'176')
```

The user-profile management commands, which enable the user to adapt or match the system to his needs during a terminal session, provide faster and easier setup of problems, and easier definition and entry of commands, operands, values, or expressions. The commands and the system functions they request are shown in Table 12.

Table 12. User-Profile Management Commands and Their Functions (The commands are listed in the same order as the command descriptions that follow later in this section.)

Command	Function
DEFAULT	Add, replace, or delete entries in default table.
SYNONYM	Rename specified command, operand, value, or expression.
PROFILE	Replace user profile in USERLIB with session profile.

SET can also be used to establish command symbol values that are part of the session profile.

User Profile

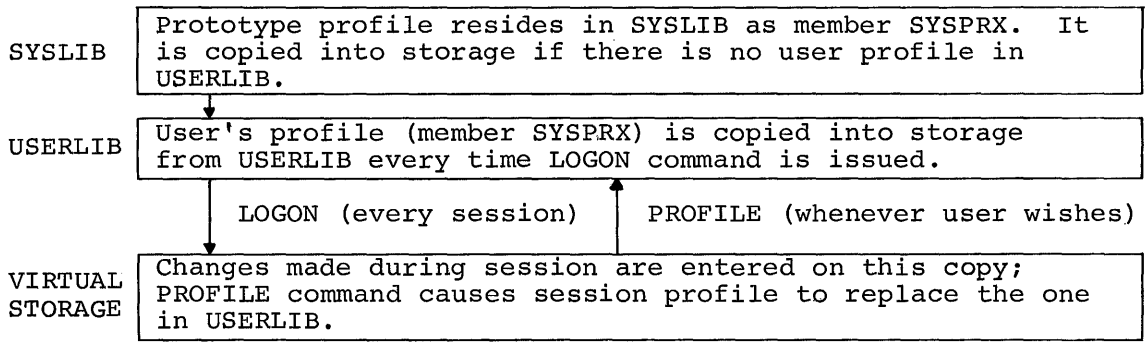
The user profile is a specialized data set containing information pertinent to each user. Stored within this data set is information regarding the values the user generates for defaults and synonyms, and (optionally) his command symbols. The user profile is a member of the user library (USERLIB).

Initially, the system provides the user with a prototype user profile (in SYSLIB) that contains the default values for system-supplied commands and any initial synonym values. The user can make changes to the prototype copy of the profile, when it is in storage; he can add to it during a terminal session by issuing a SYNONYM or DEFAULT command, or by using the SET command to establish command symbols. Such changes affect only the session profile, unless followed by the PROFILE command, which permanently changes the user's profile.

When the prototype profile is not permanently changed during a session, the copy in storage is erased when LOGOFF is issued. When, during a session, the user issues a PROFILE command, the entire profile copy in storage is written into USERLIB, and given the member name of SYSPRX.

When the user initiates his task, the system generates a search through USERLIB to locate the user's profile. If the profile is not found (i.e., the user has erased it or he did not create it), the system copies the prototype profile from SYSLIB into storage, where it may be accessed and used. Unless preserved via PROFILE, this storage copy of the prototype profile is erased at LOGOFF.

The user profile can be, concurrently, on three levels: the prototype profile in SYSLIB, the user's profile in USERLIB, and the session profile in storage.



At the user's first LOGON, the system provides initial default values for most operands. When the user does not explicity define operand values while he is entering a command requiring these values, the system will default to the initial value that it has provided. If the initial value is null, the user must specify a value. A list of default values, supplied by the system, is shown in Appendix C.

A user can specify his own default values, to be used in place of or in addition to the system-supplied default values, by using the DEFAULT command. Any changes become a part of his user profile for the session involved and may, of course, be saved for later sessions by issuing a PROFILE command.

Each user has a separate user library and, therefore, a separate user profile. Sometimes the user may find it desirable to share the copy of the profile in his user library. Since his copy is addressable as a normal member, it can be shared by making USERLIB shareable. Normal sharing precuations and procedures should be used.

The user may erase his copy of the user profile by using the normal erasing procedure.

DEFAULT Command

This command changes default values supplied by the system. Also, since some operands have the same value during one session or during successive sessions, the DEFAULT command minimizes the necessity of entering the same value several times, by assigning a value to an operand in advance of its use.

Operation	Operand
DEFAULT	{operand= [value]} [,...]

Note: Keyword operand format is not valid.

operand

designates the operand whose default value the user wants to alter or establish.

Specified as: an operand name (i.e. keyword).

value

specifies the value to be assumed whenever the specified operand is omitted in a command. This value does not apply when an operand value is explicitly given for the operand in a command in which it appears. This value overrides any previous default assigned to the operand during the task.

Specified as: a normal or quoted string.
System default: any previously assigned default value for the specified operand is omitted.

Functional Description: The system will add, replace, or delete entries in the user's default table, according to the specifications of the command. When the user has assigned a value to an operand by issuing DEFAULT, he can enter commands without explicit statement of the operand; the system will use the value he assigns.

Programming Notes: The DEFAULT command can be used to delete default entries previously defined. The user enters the DEFAULT command, and specifies a null string to be assigned to the operand name he wants to delete.

Example: The user wants to change the default value for DSNAME to A.B, which is a partially qualified data set name.

```
User:     default  dsname=a.b
          delete

System:   (deletes data sets with names starting with A.B and displays deleted names)
```

SYNONYM Command

This command renames commands, operands, values, or expressions. The renaming is valid for the remainder of the task in which it is established and can be made permanent by using the PROFILE command.

Operation	Operand
SYNONYM	{term = [character string]} [,...]

Note: Keyword operand format is not valid.

term

designates the new name of a command, operand, value, or expression.

Specified as: a normal or quoted string.

character string

specifies the value of the term that is to be used when the term is referenced; this value overrides any string value previously equated to the term.

Specified as: a normal or quoted string; maximum length, 244 bytes.
System default: any previously assigned synonym term is deleted.

Functional Description: Synonyms may be equated to other synonyms. The final line processed by the command system is the net result of all commands that might change its contents, such as SYNONYM, DEFAULT, or a procedure call.

Caution: The user should be careful to avoid excessive nesting of synonyms.

Examples:

1. The user issues a SYNONYM command.

   User: synonym a=b, b=c, c=d, d=e; a
   System: (calls procedure or program E)

   Note: Value of last SYNONYM issued overrides previous values.

2. The user executes a series of commands.

```
builtin progra
synonym pgapars = 'x,y,z'
synonym x=m
progra pgapars
```

   BUILTIN defines program PROGRA; the first SYNONYM command defines its operands. The second SYNONYM equates X and M. When the procedure is called, synonym substitution occurs, and this command is executed:

```
progra m,y,z
```

PROFILE Command

This command causes the session profile to replace the user profile in USERLIB, which will be automatically invoked whenever a task is initiated.

Operation	Operand
PROFILE	[CSW= {N|Y}]

CSW

specifies whether the command symbols are to be saved with the profile.

Specified as: Y - yes
N - no
System default: N is assumed.

Functional Description: When a PROFILE command is issued, the virtual storage profile is read into the user library, replacing the previous version, and remains unchanged until another PROFILE command is issued. As a result, values entered by DEFAULT or SYNONYM, and (optionally) SET commands are made parts of the user profile that is in USERLIB.

Programming Notes: This command is used when the user wants his current session profile to be used for all subsequent sessions.

Examples:

1. Save a profile with no command symbol definitions.

   User: profile
   System: (reads virtual storage profile into USERLIB)

2. Save a profile with command symbol definitions:

   User: profile csw=y
   System: (reads virtual storage profile into USERLIB and saves command symbols)

The user can alter command names, redefine system-supplied commands, and create new commands from a combination of system-supplied commands and/or assembler object coding. In creating a new command, the user can also define his own operands and establish the desired defaults for these operands.

To create commands, the user has two system-supplied commands, PROCDEF and BUILTIN.

- PROCDEF defines a command procedure, consisting of a combination of other commands that the user can invoke as a command.
- BUILTIN defines an object program that the user can invoke as a command.

User-written commands offer advantages to the user.

1. Although it is possible to store a series of commands (e.g., a non-conversational SYSIN data set) and then execute them via the EXECUTE command, this process has limitations in its flexibility. PROCDEF is easier to use; it does not require the user to explicitly define a data set when the commands are to be stored, and it allows easier modification of the commands.

   A user may have a series of commands that he issues many times during a session, or several sessions. He can collect these commands in sequence as a procedure, assign a name to this procedure via PROCDEF, and subsequently invoke this procedure simply by issuing the name. Since some of the commands in the procedure may require operands, provision is made by the system to associate the operands with the name of the command procedure. After the procedure has been defined and named, it may be executed by entering its name and the necessary operands during a session, just as any system-supplied command is entered. Defining a command procedure is analagous to writing a computer program; that is, a set of commands is established at one time, to be executed at a later time, by issuing its name.

2. The user may require an entirely new command which invokes actions unlike those provided by any current system-supplied commands. He creates an object program and, by using the BUILTIN command, defines his object code as a user-written command. This procedure is called by its name in the same way a system-supplied command calls a procedure. It differs from a normal object module call, however, in that operands may be supplied according to command-operand rules, rather than program-call rules.

Command Procedure

A command procedure is a prestored, parameterized sequence of command statements and other input material necessary for the execution of the statements. The user calls the procedure by issuing the procedure name as a command. For example, if he has defined a command procedure by using PROCDEF and has specified ABC as the procedure name, he may call his procedure simply by issuing ABC. The procedure call is a two-stage process. In the first stage, operand substitution is made where specified; in the second, lines of the procedure, consisting of commands, are scanned and executed in the same manner as a system-supplied command entered at the terminal.

SECTION 1: COMMAND PROCEDURE DEFINITION---PROCDEF

The PROCDEF command defines a command procedure that consists of other commands. In issuing PROCDEF, the user must specify, as an operand, the name to be assigned to the new user-written command procedure. This procedure name is the command that invokes the procedure.

Operation	Operand
PROCDEF	NAME=procedure name

name

designates the name to be assigned to the command procedure.

Specified as: one-to-eight characters; must not contain imbedded blanks, commas, semicolons, equal signs, or apostrophes.

Specifying Dummy Operands

After the user enters the PROCDEF command name and operand, the system prompts him to enter the first line by printing line number 100. For example, if COPYCAT is the name of the command procedure the user wants to define, he enters:

```
procdef copycat
```

and the system replies:

```
100 (the system assigns line number 0  to the procedure header)
```

When the user wants to build his command procedure so that he can substitute values for the operands of the constituent commands when he calls the procedure, he can assign dummy names to these operands by means of the PARAM line. When he subsequently calls his procedure, he specifies, in the operand field of his user-written command, the value for each of these dummy operands.

A dummy operand name may be specified in either of two ways.

1. A character string, which defines a positional operand, is (a) the keyword name of the dummy operand used for association with the calling parameter (the value specified in the operand field of the command calling the procedure) and is (b) the internal string for which there will be a substitution in the procedure text. The actual character string specified as the calling parameter will replace all occurrences of the dummy operand in the procedure text. Example:

   ```
   param  dsname
   ```

   When the procedure is called, the value of the first positional operand or the operand value of the keyword DSNAME in the calling command will replace the occurrences of DSNAME in the procedure text.

2. An external character string (keyword) and an internal character string. The external string is the keyword name of the dummy operand used for association with the calling parameter (the value specified in the operand field of the command calling the procedure). The internal string (to the right of the equal sign) will be replaced by a substitute in the procedure text, when the procedure is called. Example:

   ```
   param  dsname=$1
   ```

DSNAME is the external string and $1 is the internal string. When the procedure is called, the value of the first positional operand or the operand value of the keyword DSNAME in the command calling the procedure will replace $1 in the procedure text. Operand resolution is discussed in detail in Section 3, "Operand Resolution and Substitution."

The user may specify dummy operands as normal or quoted strings. A normal string is a continuous group of characters that begins with any nonblank character, except an apostrophe, and ends with the last nonblank character that is prior to a comma, equal sign, or semicolon and that is external to all pairs of parentheses in the string. A normal string may also end with the last nonblank character prior to the end of the line, if there is no continuation character. All System/360 characters are valid except commas, equal signs, or semicolons that are external to parentheses pairs in the string. For example, PARAM A+B: (C, D), A'BC''D, C, D, E contains these normal strings:

```
A + B
(C, D)
A'BC''D
C
D
E
```

A quoted string is any character string enclosed by apostrophes and within which all other apostrophes are doubled. This form provides the user with the facility to employ special characters and blanks in the dummy operand. All System/360 characters are valid. When the string is processed by the system, the terminal apostrophes are removed and the double apostrophes are replaced by single apostrophes. For example,

```
PARAM  'ABC', 'CD''D', '$2.95', 'A=B', 'GO JOE'
```

Quoted String	Internal Representation
'ABC'	ABC
'BC''D'	BC'D
'$2.95'	$2.95
'A=B'	A=B
'GO JOE'	GO JOE

Entering Procedure Text

After the user issues the PROCDEF command name and operand, and (optionally) the PARAM line, all subsequent lines issued without a preceding underscore character will be included in the procedure text. The system prompts for each line with a line number, and there is no apparent limit on the number of lines the user may enter.

The user can enter system-supplied commands (including PROCDEF and/or BUILTIN) or other user-written commands. The commands entered need not include all the operands associated with them, but only those necessary for the successful performance of the functions requested. These operands may remain variable, by specifying dummy names which also appear in the PARAM line, or may be fixed with explicit values. Fixed operand values are not included in the PARAM line and will be acted upon as specified in the text when the procedure is called.

A direct call to an object module may be entered by using the name of the module in the procedure text.

Commands preceded by an underscore are executed immediately and do not become part of the procedure text.

Terminating Procedure Definition

The user terminates PROCDEF processing by entering an underscore followed by either an END command, another PROCDEF command, or an EDIT command. When the user enters another PROCDEF command, the same options for terminating its processing are applicable; the last PROCDEF will have to be terminated with either an END or EDIT command.

Here are examples of PROCDEF usage and format.

Example 1:

```
procdef copycat
100 param ddname=alphname,dsname=name1,volume=any,$n,state=$1,$2
200 ddef ddname=alphname,dsorg=vi,dsname=name1,volume=any,$n
300 catalog dsname=name1,state=$1,$2
400 _end
```

In the PARAM line, DDNAME, DSNAME, VOLUME and STATE are external strings (keywords) that associate the calling parameters with the internal strings (in the PARAM line): ALPHNAME, NAME1, ANY and $1, respectively. These internal strings will be replaced in the procedure text by the calling parameter values. $N and $2, represented positionally in the PARAM line, will also be replaced by a substitute in the text.

DDEF, on line 200, is a system-supplied command, with the variable operands DDNAME, DSORG, DSNAME, VOLUME and DISP. The keyword DISP is omitted and the dummy operand $N is supplied positionally. DSORG=VI is a fixed operand value and will be acted upon as specified. Values for the other variable operands will be supplied when the procedure is called.

CATALOG, on line 300, is also a system-supplied command. Its operands are variable and will be replaced by substitutes when the procedure is called. When END is entered, the definition of this procedure is terminated.

Example 2:

```
procdef  dmprog
100  param $1, 'here:  there'
200  if '$1'='yes'; display 'success'
300  if '$1'¬='yes'; dump here:  there
400  _edit xyz
```

The quoted string 'HERE: THERE' is given as a dummy operand in the PARAM line. The apostrophes permit the use of the colon within the character string, although the apostrophes will be removed when the string associated with this dummy operand is substituted in the procedure text.

In lines 200 and 300, IF is a system-supplied command; the apostrophes enclosing its operands will not be removed when the substitution is effected.

_EDIT xyz, in line 400, is used to terminate the PROCDEF.

Example 3:

```
procdef diff
100  param alphname, $1, name1
200  ddef  ddname=alphname,dsorg=$1,dsname=name1
300  _procdef callme
100  param alphname,$A, $1, $2, '$3', $4
200  asm name=alphname, macrolib=$a,y,y,y,y
```

```
300  copycat alphname, $1,$2,$3,$4
400  _procdef ditto
100  param myprog
200  myprog
300  _end
```

The underscore character preceding each PROCDEF command (after the first) indicates that a series of procedures are defined. Each succeeding PROCDEF terminates the execution of the PROCDEF preceding it. To end the series of PROCDEFs, _END is used. Within CALLME (line 300), the user-written command COPYCAT is used, since it was defined by a PROCDEF. In the last PROCDEF, MYPROG will be executed as a direct call when DITTO is invoked and the name of MYPROG is supplied.

Nested PROCDEFs

The text of a procedure, defined by PROCDEF, may contain other PROCDEF commands, entered as any other system-supplied command (i.e., without a preceding underscore). These are termed nested PROCDEFs. Here are some of the uses of nested PROCDEFs.

Example 1:

```
procdef  abc
100  param $1,alphname,dsname=name1
200  ddef alphname,vi,nameldisp=new
300  catalog dsname=name1,state=u,newname=alphname
400  procdef $1
500  param $2, name2, dsname=name3
600  ddef name2,vs,dsname=name3
700  default sysinx=e
800  procdef $2
900  myprog
1000 ___end
1100 __end
1200 _end
```

In defining procedure ABC, lines 100 through 1100 are treated as text of ABC. When END, preceded by three underscores on line 1000, and END, preceded by two underscores on line 1100, are entered, the first underscore in each case is ignored and dropped since it is immediately followed by another underscore. The remaining underscores are entered as text of the line. The END preceded by a single underscore on line 1200 terminates the execution of PROCDEF ABC.

When the command ABC is entered, the values of the calling parameters are substituted for the occurrences of the dummy operands throughout the procedure. The procedure is then executed so that the commands DDEF and CATALOG will be executed. When the first PROCDEF command is encountered (as the third command of procedure ABC) a new procedure is defined. For example, suppose command ABC is entered like this:

```
abc one, myprog, dsname=mylib
```

Then the new PROCDEF becomes

```
PROCDEF ONE
```

The remaining lines of ABC (500-1000) form the text of the procedure called ONE, which now appears as:

```
PROCDEF ONE
100  PARAM $2,NAME2, DSNAME=NAME3
200  DDEF NAME2, VS,DSNAME=NAME3
300  DEFAULT SYSINX=E
400  PROCDEF $2
```

```
500  MYPROG
600  __END
700  _END
```

To effect the entry of the data for the PROCDEF in line 400, from the procedure defined above, SYSINX must have a value of E.

The first underscore in line 600 is ignored and the text for line 600 is stored by the system as _END. The END in line 700, preceded by one underscore, terminates the execution of PROCDEF ONE. When procedure ONE is called (and, of course, it cannot be called before ABC is called) the DDEF command will be executed and then a third procedure will be created with a name given by the first supplied operand in command ONE (i.e., the call to procedure ONE).

Note: When command ABC is subsequently entered with the first operand of ONE, procedure ONE will not be recreated since it is already in the procedure library. Instead, the text editor (which is invoked by the PROCDEF processor) will start adding lines at the end of the old procedure (see "Editing Procedures," below).

Example 2:

```
procdef  def
100  param $1,$2 alphname,dsname=$3
200  ddef alphname, vi, dsname=$3
300  default sysinx=e
400  procdef $1
500  param dsname=$3,$4
600  catalog dsname=$3,state=u, newname=$4
700  __procdef $2
800  param $a,$b,$c
900  if '$a'='yes'; display loc$b
1000 if '$a'¬='yes'; display loc$c
1100 __end
1200 _end
```

The first underscore on line 1100 is ignored and the second is entered as text. PROCDEF $1 and PROCDEF $2 are both nested within procedure DEF, not one within the other as in the example 1. When DEF is called, both PROCDEFs ($1 and $2) will be executed and the two resulting procedures will be stored. For example, DEF is called:

```
def two,four,myprog,dsname=myjob
```

The substitution of the values that are passed as the calling parameters is made for the dummy operands, and the commands are executed. PROCDEF TWO is executed and is terminated by _PROCDEF FOUR, which is terminated by END. The resulting text is:

```
PROCDEF TWO
100  PARAM DSNAME=$3,$3
200  CATALOG DSNAME=$3,STATE=U,NEWNAME=$4
300  _PROCDEF FOUR
100  PARAM $A,$B,$C
200  IF '$A'='YES'; DISPLAY LOC$B
300  IF '$A'¬='YES'; DISPLAY LOC$C
400  _END
```

Nested Procedures

Nested procedures are user-written commands that call procedures (defined by either PROCDEF or BUILTIN) within the text of a procedure.

A nested procedure may include another user-written command that calls a procedure. In each case, when a new procedure is called, it is processed before returning to the procedure from which the call was made. For example,

```
procdef tab
100  param alphname, $n, myprog, $a,$b,$c,$1,$2
200  ddef alphname, $n, dsname=myprog
300  one $a,$b,$c
400  two dsname = $1, $2
500  _end
```

TAB is then called:

```
tab datadf01, vi, data01, mycall, data03, dat03, newone, n
```

The text of the procedure then looks like this:

```
DDEF DATADF01, VI, DSNAME = DATA01

ONE MYCALL, DATA03, DAT03

TWO DSNAME=NEWONE, N
```

When TAB is executed, the DDEF command is executed and ONE is recognized as a user-written command, with MYCALL, DATA03, and DAT03 as its operands. ONE invokes its procedure and if it, in turn, calls another procedure, that call is processed before returning to process TWO, also a user-written command.

Sharing User-written Commands

User-written commands can be shared, when the owner makes his user library available to other users via the PERMIT command. The prospective sharer issues the SHARE command, with these operands: the name by which he will refer to the owner's user library, the owner's user identification, and the name of the data set to be shared (i.e., USERLIB). For example:

```
share    lib,user345,userlib
```

is the command issued where LIB is the name by which the sharer will refer to the owner's user library. Then the sharer issues a PROCDEF command, with the name of the command he wants to share as the operand. When the system prompts with line 100, he enters an underscore character followed by the EXCERPT command. He specifies, as operands, the name by which he refers to the owner's user library (in the above example, LIB), the member name SYSPRO, and the range of lines from 100 to the last line in the owner's procedure. The entire text of the procedure will be inserted into the sharer's user library.

For example, a user wants to share a command, ABC, from a user library to which he has been granted access.

After issuing the SHARE command, as above, this PROCDEF is entered:

```
User:     procdef abc

Sys,User: 100_excerpt lib.syspro,100,700

User:     _end
```

ABC will now be defined as a command in the sharer's user library and may subsequently be called by him.

Editing Procedures

The PROCDEF command invokes the text editor, enabling the user to use any of the text editing commands, while he is defining a procedure or after he has defined it, by entering an underscore character, followed by the command. He does not need to use the EDIT command.

The CORRECT command can be used within a line to respecify characters which the user may want to insert, replace, or delete. Other commands, such as INSERT, EXCISE, and REVISE, can be used to insert, replace, or delete complete lines in the procedure text. (See the commands in Part III, Section 2, "Text Editing" for a full explanation of their usage.)

This example shows how editing commands can be used during the process of defining a procedure:

```
User:      procdef      changeit

Sys,User:  100 param    name1, vi, old, name3
           200 ddef     ddname=name1, dsorg=vi, name2, disp=old
           300 ddef     name3,vi,name4,disp=old
           400 datalog  name2, state=n, access=u
           500 _correct 100

System:    PARAM NAME1, VI, OLD, NAME3

User:      *                          $2

Sys,User:  _excise 300
           _insert 400
           500 default sysinx=e
           600 edit name1
           700 _end
```

While defining CHANGEIT, the user enters text on lines 100, 200, 300, and 400. When the system prompts with line 500, the user decides to make a correction in the PARAM line. He enters an underscore character followed by the CORRECT command and the number of the line (100) he wants to modify. The system responds with the line text and the user now can enter the necessary correction characters, which specify the modifications he wants to make. The asterisk in the first column duplicates that column and all following columns until another correction character ($) is encountered. The user wants to change NAME3 in the PARAM statement to NAME2, and so, he places a $2 under the last two characters in the line. This duplicates the column above the $ and replaces the 3 with a 2.

Then, the system prompts with an underscore (rather than a line number) indicating that another command statement must be given, if processing is to continue. The user wants to delete line 300 from the procedure text. He enters the EXCISE command and line number 300. Once again, the system prompts with an underscore, indicating the completion of the command's execution and requesting the next statement.

To continue entering text, the user enters an INSERT command followed by the number of the last line entered as text, which, in this example, was line 400. Then the system prompts with line number 500 and the user enters two additional statements in lines 500 and 600. The procedure definition is terminated with the END command.

Another use of the INSERT command is shown in this procedure definition:

```
procdef    pdef
100  param alphname, vi, name1, old, name2, name3
```

```
200  ddef dsname = alphname, vi, ddname = name1, old
300  _ddef dsname = myprog, vs, ddname = test, disp = new
     _insert
300  ddef dsname = name2, vi, name3, old
400  _end
```

The user, after entering line 200, decides to issue a command statement to the command system. When the system prompts with line 300, the user enters an underscore character followed by a DDEF command, which will not become a part of the procedure. When the DDEF is completed, the system prompts with an underscore character, requesting the next command statement. The user enters an INSERT command with no line number specified; and the system prompts with the line number specified by the current line pointer, in this case line 300. This use of the INSERT command is possible because the current line pointer was not changed by the issuance of an editing command for a previous line number. The user continues to add to the procedure text before terminating PROCDEF processing with the END command.

When a procedure has been defined, the user may enter the PROCDEF command followed by the name of the procedure he wants to modify. The text editor is invoked and the system responds by prompting with the next available line number in the procedure.

For example, the user again wants to modify the procedure CHANGEIT, which he has previously defined. He enters:

```
procdef changeit
```

and the system prompts with the next available line number in the procedure; in this case, line 700. The user may now enter any of the text editing commands, preceded by an underscore character, or he may add to the procedure text by entering the desired text.

To delete a procedure that has been defined, the user enters the PROCDEF command followed by the name of the procedure; then he issues the EXCISE command, specifying the range of lines of the procedure as operands. For example, to delete the previously defined procedure, CHANGEIT:

```
User:     procdef    changeit
Sys,User: 700_excise  0, last
```

EXCISE deletes the entire procedure from line 0 to the last line.

Note: Line 0 must be specified, since it is the line number assigned by the PROCDEF command to the procedure header.

Interrupting Procedure Definition

When the user is defining a procedure and wants temporarily to interrupt PROCDEF execution to invoke the command system, he may do so by pressing the ATTENTION key at his terminal. The system responds with an exclamation point or a logical-NOT sign. At this time the user may issue any commands except PROCDEF and EDIT. When he has completed his interruption process and wants execution to continue, he must issue the GO command. The text editor positions itself to the beginning of the last line interrupted, no matter at what position within that line the interruption was acknowledged. Any data entered on this line, before the attention interruption, is lost; only the line number is saved.

Caution: If PROCDEF or EDIT are issued during the interval after the attention interruption and before the GO command, the partially completed procedure is stored in its current state.

Prompting During Execution

During a procedure call, the system may have to prompt for information; the system will expect a reply from the source of input. In conversational mode, the system will go to the terminal or the PROCDEF data set for the reply; in nonconversational mode, the system will go to the SYSIN data set or PROCDEF data set. Table 13 lists the commands for which the system will go to the terminal or the card reader for the reply. For all others, the SYSIN macro instruction is the source of input. For example, if the user were to stipulate in a PROCDEF that he wanted to assemble a data set that is not stored, the system would go to the terminal or card reader for the source data set, not to the procedure. In conversational mode, when using the assembler, the system would also issue prompting messages to the terminal.

Table 13. Source of Input: Terminal or SYSIN Data Set

ASM	DATA	FTN	PERMIT	SHARE
BACK	DDEF	LINE?	POD?	TIME
CANCEL	DELETE	LNK	PRINT	TV
CATALOG	DSS?	LOGOFF	PUNCH	VT
CDD	ERASE	LOGON	RELEASE	VV
CDS	EXECUTE	MODIFY	SECURE	WT

Diagnostic Messages During Execution

Diagnostic messages that occur during the execution of a command procedure will be output to SYSOUT, which may be a data set or terminal depending on the mode of operation. When the diagnostic message requests the user to repair an error condition, the user can make the correction at his terminal and the procedure will continue executing. If the error is nonrecoverable, the diagnostic message will be output to SYSOUT. In nonconversational mode, the task will be terminated; in conversational mode, the user will be queried for a new command.

SECTION 2: OBJECT PROGRAM DEFINITION---BUILTIN

This command defines an object program that the user can invoke as a command. It is useful for accomplishing actions not achieved by any current system-supplied commands or combination of them. The user creates an object program and defines it as a command by use of BUILTIN.

Operation	Operand
BUILTIN	NAME=command name [,EXTNAME=BPKD macro name]

NAME

designates the name of the command that calls the object program.

Specified as: one-to-eight characters, not containing imbedded blanks, commas, semicolons, equal signs, or apostrophes.

EXTNAME

the external symbol assigned as the name of the BPKD macro instruction (BUILTIN procedure key definer); see Assembler User Macro Instructions. This name becomes the external name of the called program and is the link needed between the command and the routine to be called.

Specified as: one-to-eight alphameric characters, the first of which must be alphabetic.
System default: value given in NAME is assumed.

The user, as with a PROCDEF, can define operands and supply operand values when his user-written command is issued.

If the user wants to define operands for his command, he must supply the coding within his module to handle the parameter values supplied when the module is called. The BPKD macro instruction must be supplied in the object code as part of the PSECT and must include the definitions of the expected parameters. The macro instruction must also supply the names needed to provide linkage between the module and the BUILTIN command that defines that module. Refer to Assembler Programmer's Guide for a further description.

SECTION 3: OPERAND RESOLUTION AND SUBSTITUTION

The user can specify operands for user-written commands, created with either the PROCDEF or BUILTIN commands. With PROCDEF, the user is primarily establishing the operand values that are required as parameters by the commands that constitute the command procedure. He specifies these parameters by entering, on the line following the PROCDEF command, the word PARAM followed by the dummy names he wants to specify for the operands within the commands that constitute his procedure.

When the user wants to define parameters for a BUILTIN procedure, he must supply the coding within his module to handle the parameter values supplied as operands. He must also provide a BPKD macro instruction within the module to generate the linkage to the object program defined by BUILTIN. Pointers are then generated to specify the address within the module where the operand names are stored. When the user issues the command, any operand value given with the command is passed to the module by pointers in the locations provided by the BPKD macro instruction.

With both types of user-written commands, the parameters supplied as operands are resolved at the time the user-written commands are issued. The resolution is identical to that given parameters for system-supplied commands.

For commands created with PROCDEF, the system goes through a procedure-expander routine that performs two functions: operand resolution and operand substitution. Operand resolution consists of:

1. Analyses of calling operands
2. Analyses of procedure operands
3. Generation of operand equivalences

Operand substitution, as performed in the expansion of a procedure, has no counterpart in operand resolution of a command created with BUILTIN, and has a significant effect on the final evaluation of a procedure. Operand substitution is explained later in this section.

Analyses of Calling and Procedure Operands

When the user calls a procedure, he enters the command (the procedure name) followed by the operand values he wants assigned to the dummy operands of the procedure. As with system-supplied commands, operands may be represented either positionally or by means of a keyword.

Positional notation: Positional calling operands must be supplied by the user in the same order as that given in the procedure parameter list (PARAM line) or the BPKD parameter list. When a positional operand is omitted and another positional operand is written following the omitted operand, the comma that would have followed the omitted operand must be retained to indicate the relative position of the operand that is included.

Assume this command procedure defines a new or existing data set whose organization is always VISAM.

```
PROCDEF VIDDEF
100 PARAM DDNAME, DSNAME, DISP=NEW
200 DDEF DDNAME, VI, DSNAME, DISP=NEW
300 _END
```

This procedure call might be used:

```
viddef mybest, testl, old
```

VIDDEF is the command that calls the procedure. The first positional operand in the calling sequence is MYBEST, which is the value assumed by the first positional dummy operand (DDNAME) in the PARAM list. TEST1 is the value assumed by DSNAME, both being positionally related, (i.e., second position). The third positional operand in the calling sequence is OLD, which is the value assumed by the third positional operand in the PARAM line, even though this dummy operand is shown in keyword notation. OLD replaces the dummy value NEW in the keyword notation. The result of the above procedure call is:

```
DDEF MYBEST, VI, TEST1, DISP=OLD
```

For the same procedure, assume this procedure call:

```
viddef mybest, vs, testl, old
```

This procedure call is erroneous. The user wanted to specify operands to be inserted directly in the DDEF command in the procedure, to change the data set organization to VSAM. However, by positional association of the calling operands and the PROCDEF PARAM line, these associations are made:

```
DDNAME = MYBEST
DSNAME = VS
DISP = TEST1
```

OLD, which has no positional relationship, is ignored and a diagnostic message may be issued to SYSOUT.

The result of the above procedure call is:

```
DDEF MYBEST, VI, VS, TEST1
```

MYBEST is the data definition name, VI is still the data set organization, VS is the data set name, and TEST1 is the disposition. There could be a data set whose name is VI, but DISP must be OLD or NEW, not TEST1.

Keyword notation: Keywords of calling operands may appear in any order; of course, each keyword has an associated positional notation. Keywords have the general form: KEYWORD=value, where KEYWORD is the name of the operand, followed by an equal sign, and value is the actual value of the operand.

Keyword and positional representation may be used simultaneously in a string of operands. When one operand is expressed in both manners, the occurrence that appears last (left to right) in the string is assumed for the value.

Assume this procedure has been defined to define a data set to dump one or more data locations or expressions:

```
PROCDEF AUTODUMP
100 PARAM DSNAME = ALPHNAME,DISP = NEW, DATA = 'HERE:THERE'
200 DDEF DDNAME = PCSOUT, DSORG = VI, DSNAME = ALPHNAME, DISP = NEW
300 DUMP HERE:THERE
400 _END
```

This procedure call might be used:

```
autodump data = '0:15r, 0:6d, top: middle', old, dsname = myprog
```

This call has a combination of keyword and positional notation, and the keyword notation does not coincide with the corresponding positional notation in the PARAM line of the procedure. The result of the above procedure call is:

```
DDEF DDNAME=PCSOUT, DSORG = VI, DSNAME = MYPROG, DISP = OLD
DUMP 0:15R, 0:6D, TOP:MIDDLE
```

The above example shows how a quoted string may be used to define several operands within a single operand field. Although an operand in the DUMP command is not shown in keyword notation, it may be designated as keyword notation in the PARAM line and calling sequence. The keyword value 'HERE:THERE' in the PARAM line must be identical to the value expressed as an operand in the DUMP command (i.e., HERE:THERE). The user could have used FROM:TO or FIRST:LAST or any expression as an operand of the DUMP command, but the keyword value in the PARAM list must be identical to the DUMP operand expression. The apostrophes are needed in the keyword expression only if embedded blanks or special characters are used within the keyword expression.

Defaults: The user can specify, alter, or delete default values for the operands of a user-written command in the same way as with a system-supplied command. The operands to be specified with the command are mandatory, unless the user has provided defaults; he must supply mandatory operands every time he issues his command. The user creates a default value by issuing the DEFAULT command (see Part V, "User Profile Management"), specifying the dummy operand name in the operand field.

If the user invokes his command procedure and omits operands, the system either obtains the default value (if one exists) from the user library, or substitutes a null string for the missing value.

For an object program defined with BUILTIN, the user can write a routine to either generate a message or supply a fixed value when a mandatory operand is omitted. For example, this command procedure has been defined:

```
procdef  defcat
100 param ddname = datname, dsname = alphname, dsorg = vp
200 ddef ddname = datname, dsorg = vp, dsname = alphname
300 _end
```

DEFCAT is used to define a data set. This information should be noted about the DDEF command:

1. The default value for DISP will be assumed, since this operand is not given.

2. A keyword value of VP, which is not the default value of DSORG (VI), is assigned to keyword DSORG.

Assume this sequence of commands:

```
default disp = old, dsorg = vs
defcat textx, dsname = prog1, ddname = textxy
```

The result of that sequence is:

```
DDEF DDNAME = TESTXY, DSORG = VS, DSNAME = PROG1, DISP = OLD
```

Explanation:

1. DDNAME = TESTXY

   Although the dummy operand DDNAME = DATNAME is in keyword notation, it can also be considered positionally (i.e., first operand in the PARAM line). The system resolves TESTX as a possibility of being the value for DATNAME. However, DDNAME = TESTXY is specified elsewhere in the calling sequence and the system now has another possibility (TESTXY) for a value of DATNAME. The system always assumes the last appearance, from left to right, of a multispecified operand.

2. DSORG = VS

   The system-supplied default value for DSORG is VI. DSORG = VP is given as an operand in the DDEF command, and in the PARAM line. The DEFAULT command defaults DSORG to VS. As a result of the procedure call, no indication is given for the value to be substituted for VP, either in positional or in keyword notation. The system searches for a default value: It checks the user default table first, then the system-supplied default table. The user has given a default value for DSORG and this value (VS) is assigned by the system as a string to be substituted for VP in the PARAM line. Eventually the system will substitute VS for the DDEF operand keyword value, VP.

3. DSNAME = PROG1

   This operand is given in the procedure call in keyword format. The keyword value PROG1 replaces the keyword value ALPHNAME in the PARAM line; also, PROG1 is substituted as the keyword value for every occurrence of ALPHNAME in the body of the procedure.

4. DISP = OLD

   In the DDEF command in the procedure text, the dummy operand DISP has been defaulted, so it does not appear in the PARAM line. The system default for DISP is NEW; however, the user has issued his own default for DISP (i.e., DISP = OLD), which takes precedence over the system default.

   Any other mandatory operands that were omitted would assume system default values.

Nulls: The user can specify a null value (absence of a value) for the operand of a user-written command; a null value is a quoted string of 0-length. Indicating a null value for an operand has an effect that is different from indicating a default value (i.e., absence of an operand). A null value for an operand can be achieved in these ways:

Assume operands A=x, B=y, C=z

1. By omitting keyword operand A=x and specifying another operand in its position, a default value will be assumed for A. If there is no default value, a null value is assumed.

   ```
   B=y, C=z
   ```

2. A null value for A can be expressed by the use of two successive apostrophes in the position of operand A.

   ```
   '', B=y, C=z
   ```

3. Keyword notation can be used to indicate a null value for A.

```
A='',B=y,C=z
```

Here is an example of the use of a null value:

```
procdef copy
100 param dsname1, dsname2, base, incr
200 if 'base' = ''; cds dsname1, dsname2
300 if 'base'¬ = ''; cds dsname1, dsname2, base, incr
400 _end
```

Calling sequence 1:

```
copy orig, dupe, base = ''
```

Assume the user has previously specified a default value of 300 for BASE. The data set names ORIG and DUPE will be substituted in lines 200 and 300 of COPY. BASE is indicated with a null value, which is assigned to the dummy BASE operand in the PARAM line. The null value will also be substituted wherever the character string BASE appears in the text of COPY, as shown below.

```
200 IF '' = ''; CDS ORIG, DUPE
300 IF ''¬= ''; CDS ORIG, DUPE, '', 100
```

After substitution, the quoted string 'BASE' becomes a quoted string with no space between the apostrophes since a null value is actually a quoted string of 0-length. The system default value of 100 is assumed for the operand INCR, in line 300, since INCR was defaulted in the calling sequence.

For calling sequence 1, therefore, the conditions are met in the conditional statement in line 200; the associated CDS command will be invoked.

Calling sequence 2:

```
copy orig, dupe
```

Assume, again, that the user has previously specified a default value of 300 for BASE. BASE and INCR are defaulted in the calling sequence. The user-supplied default value of 300 for BASE and the system-supplied default value of 100 for INCR will be the values assigned to the operands BASE and INCR in the PARAM line. The result of the substitution process is:

```
200 IF '300' = ''; CDS ORIG, DUPE
300 IF '300'¬ = ''; CDS ORIG, DUPE, 300, 100
```

The conditions for the conditional statement in line 200 are not met (i.e. '300' does not equal '') but the conditions for the conditional statement in line 300 are met, so that CDS command will be executed.

Synonyms: The user can specify synonyms for his dummy operand names by using the SYNONYM command (see Part V, "User Profile Management").

Since the user can specify synonyms, defaults, and nulls for operands, the system must have a sequence for resolving these values.

1. Synonyms are checked first. For example, if this series of synonyms has been set up:

```
A = B
B = C
C = D
```

then the final value for either A, B, or C is D. Operands are searched from left to right until the operand list is exhausted. For a given synonym, the value is the total string read to the next semicolon or to the end of an unhyphenated line.

2. If an operand has no synonym, the system searches both the user- and system-default tables. If no default value has been assigned, the value of the assumed operand is null.

Example: This procedure has been defined.

```
procdef example
100 param $a, $b, $c, $d, $e, $4
200 if $a = $b; display $c, $d, $e, $4
300 end
```

Later, the user issues a command statement.

```
default $d = '01:5r', $e = '0:6d', $4 = 'top:bot'
synonym $c=$b, $a=$b
```

Then, he calls his procedure.

```
example $b=loc50
```

The following steps, taken by the system, are internal and are shown here for clarification only. Each operand is checked first for synonyms and then for defaults.

```
$A = $B (SYN) = LOC 50 (value in calling parameter list)
$B = LOC50 (value in calling parameter list)
$C = $B (SYN) = LOC 50 (value in calling parameter list)
$D = 0:15R (default value)
$E = 0:6D (default value)
$4 = TOP: BOT (default value)
```

After final resolution and substitution, a new procedure is produced.

```
IF LOC50 = LOC50; DISPLAY LOC50, 0:15R, 0:6D, TOP: BOT
```

Generation of Operand Equivalences

The system establishes a table of dummy operands and their corresponding specifications, and accounts for positional and keyword operands and dummy operands. Also, when the system recognizes that two specifications are made for the same operand, a diagnostic message will be issued.

The result generated is shown in Table 14. The first column, INTERNAL STRING, contains the character string that is the dummy operand keyword or positional value in the PARAM line, which will be substituted within the body of the text wherever an identical string occurs. The second column, KEYWORD, contains the dummy keyword operand in the PARAM line. The third column, VALUE, contains either the keyword or positional value expressed in the calling sequence. When a call is made on a procedure, the keyword column is searched for each calling parameter keyword. If one is found, the value associated with the calling keyword is placed in the VALUE column; this value will be substituted later for the associated string in the INTERNAL STRING column.

Table 14. Generation of Operand Equivalences

INTERNAL STRING	KEYWORD	VALUE
Substitute string in body of text	Keyword in PARAM list	Keyword value in calling sequence

Here is an example of how resolution of operands occurs, based on the process shown in Table 14.

Assume this procedure has been defined:

```
procdef asmwlist

100 param alphname, stored = $n, lincr = (first,last),-
   version, symlist = $y

200 asm alphname, stored = $n, lincr = (first,list)

300 verid = version, isd = y, symlist = $y, asmlist = y,-
   crlist = y, stedit = y, isdlist = y, pmdlist = y

400 _end
```

This procedure call is then made:

```
asmwlist myprog, stored = y, version = today, now,-
   alphname = myprog 1, symlist = n
```

The effect of operand resolution is indicated in Table 15.

Table 15. Indication of Operand Resolution

POSITION	1	2	3	4	5
PARAM string	ALPHNAME	STORED=$N	LINCR=(FIRST,LAST)	VERSION	SYMLIST=$Y
Calling values	MYPROG1	STORED=Y		NOW	SYMLIST=N
String for which substitution will take place	ALPHNAME	$N	(FIRST, LAST)	VERSION	$Y

For each string named, a value will be ascertained.

Internal String	Keyword	Substitute Values
ALPHNAME	ALPHNAME	MYPROG/MYPROG1
$N	STORED	Y
(FIRST,LAST)	LINCR	null
VERSION	VERSION	TODAY/NOW
$Y	SYMLIST	N

The last value in each line will be taken, so that the system's table of operand equivalences would look like this:

Internal String	Keyword	Value
ALPHNAME	ALPHNAME	MYPROG1
$N	STORED	Y
(FIRST,LAST)	LINCR	null
VERSION	VERSION	NOW
$Y	SYMLIST	N

Operand Substitution

After resolution of operands, as described above, a substitution process occurs.

The result of the procedure expansion, after operand substitution, would be:

```
ASM NAME = MYPROG1, STORED = Y, LINCR = , VERID = NOW, ISD = Y,
   SYMLIST = N, ASMLIST = Y, CRLIST = Y, STEDIT = Y, ISDLIST = Y,
   PMDLIST = Y
```

Note: In the PARAM line, the keyword value of SYMLIST was specified as $Y; the $ ensures that the calling sequence keyword value (N) associated with SYMLIST would be substituted only where the string $Y occurs in the body of the text. For example, if the dummy operand were SYMLIST = Y, then, for every occurence of string Y, string N would be substituted.

The first letters of each dummy operand keyword value name in the PARAM line, beginning with the last operand, are gathered into a string. Every character of the procedure text is compared with the characters in this string. If a matching character is found, the remaining characters of the particular dummy operand, of which the matching character was the first, are compared with the succeeding characters of the procedure line. Note that the "characters of the procedure line" include all characters, whether they are characters of a command, operand, or comment within the body of the procedure.

When a full match (i.e., an entire dummy operand is matched) is found, the characters of the procedure line are replaced by the calling parameter keyword value. If not found, the remaining characters of the string, consisting of the other first letters of dummy operands, are compared with the procedure line, as above. If a character in the procedure text is not the same as the starting character of any of the operands, no substitution is made for that character. The comparison continues with the next character in the procedure text until all characters in the procedure have been compared.

If there are no variable operands, the procedure text remains intact and no substitution will take place.

Examples:

1. Assume a user has defined three procedures, PROCLOAD, PROCRUN, and PROCTEST, to perform three different functions. Each has its own set of operands, which may or may not be similar in both name and number. The user would like to be able to call any of these procedures by use of a single name, using a new procedure.

```
procdef procall
100 param op, funct, 'p' = list
200 opfunct list
300 end
```

Later, he issues this procedure call to invoke PROCLOAD.

```
procall proc, load, 'my progx, 1.0, h''50'''
```

After operand resolution occurs, substitution takes place. The first letters of the dummy operand string are gathered into a string (LFO); every character in this string is compared with the characters in the procedure text. The characters in the string LFO are first compared with line 200. When the L is matched in line 200, the remainder of the dummy operand, of which L was the first letter, is compared with the characters following the L in line 200. A full match is found. The same results occur when F and O are compared with line 200. For every full match, the calling sequence keyword value replaces the corresponding set of character strings in the procedure text.

	OP	FUNCT	LIST
are replaced by	PROC	LOAD	MYPROGX, 1.0,H'50'

Following this substitution, the procedure is executed and the result is a new procedure call to PROCLOAD, with the calling parameters: MYPROGX, 1.0,H'50'.

2. This example will show the effects if dummy operand names are carelessly selected. Assume this procedure has been defined:

```
procdef starter
100 param  progname, pari=a, b, c, r
200  abacus, name=progname, a, b, c, r
300 _end
```

Later, a procedure call is issued.

```
starter zap, 1.0, c=first, r=loca
```

After operand resolution and substitution, the result is undesirable.

```
1.01.0FIRSTUS NLOME=ZAP,1.0,,FIRSTLOCA
```

The PARAM line should have unique character strings for dummy operands. The result of substitution would have been correct if the procedure had been written like this:

```
procdef starter
100 param progname, par=$a, $b, $c, $r
200 abacus name=progname, a, b, c, r
300 _end
```

The procedure call is issued as before.

```
starter zap, 1.0, c=first, r=loca
```

The result would now be

```
ABACUS  ZAP,  1.0,  FIRST,  LOCA
```

3. This example will illustrate that substitution occurs on complete matches with operands in a right-to-left occurrence. Assume that two procedures have been defined.

```
PROCDEF FAKE1                    PROCDEF FAKE2

100 PARAM AB,ABC,ABCE,           100 PARAM ABCE,ABC,AB,C,E
   C,E

200 IF 'AB'='ABC';               200 IF 'AB'='ABC';DISPLAY ABCE
   DISPLAY ABCE

300 _END                         300 _END
```

The calls made are

```
fake1 loca, no, cond,-           fake2 loca,no,cond,c='',e=code
   c='',e=code
```

After operand resolution and substitution,

```
FAKE1                            FAKE2

IF'LOCA'='NO';DISPLAY            IF'COND'='COND ';DISPLAY COND
   COND                             CODE
```

Note that these two procedures differ only in the order in which the dummy operands are specified. In FAKE1 the system found a match for ABC before it found one for AB; the situation is reversed in FAKE2.

SECTION 4: PROCDEF EXAMPLES

The user is expected to make extensive use of the facilities with which he can create his own commands, primarily by the use of the PROCDEF command. The user can save time by combining a frequently used series of commands into one command. The series of examples, below, illustrate PROCDEF usage and applications that achieve greater efficiency.

1. The user wants to combine the DISPLAY and DUMP commands so that he has only one command, with the option of displaying data at the user's terminal or at the printer. Also, if DUMP is used, an automatic DDEF command is generated, defining the data set to be dumped.

```
PROCDEF     OUTPUT

100 PARAM   ALTER, DATA1, DATA2, DATA3, DSNAME

200 IF 'ALTER' = '' | 'ALTER'='Y'; DISPLAY DATA1, DATA2, DATA3


300 IF 'ALTER'='N'; DDEF PCSOUT, VI, DSNAME: DUMP DATA1-
       DATA2, DATA3

400 _END
```

ALTER serves as a switch; if 'Y' is specified or ALTER is omitted, DISPLAY is executed; if 'N' is specified, DUMP is executed. The user may execute a DUMP to display two data fields at the printer with this calling procedure:

```
output alter=n, data1=field1, data2=field2, dsname=data set name
```

Only line 200, in the above procedure, is executed; this causes a DDEF to be issued and the two fields to be dumped.

To display one data field, the user issues

```
output data1=field1
```

Only line 100 is executed. DSNAME need not be specified since the DDEF is not executed.

2. The user wants to have EDIT and REGION issued automatically every time he uses a text editing command. As a result, the user does not have to invoke the editing facilities by issuing an EDIT command each time, nor does he need to issue separate REGION commands. He also wants to retain the names of the editing commands. He must define a procedure for each command involved but, for illustrative purposes, only the UPDATE command is shown.

```
PROCDEF CHANGE
100 PARAM DDNAME,RNAME
200 EDIT DDNAME
300 REGION RNAME
400 UPDATE
500 _END
```

If the user wants to update a region in a data set, and the text editor is not invoked, he issues the CHANGE command. For example, to update a data set with a DDNAME of MYDATA in the region XYZ, the user enters

```
change ddname=mydata, rname=xyz
```

3. Rather than issue a separate PROFILE command whenever he wants a synonym or default to be made part of his user library, the user causes the PROFILE command to be an option of either SYNONYM or DEFAULT.

```
PROCDEF DEF
100 PARAM PAR, SPEC, CSW, SAVE
200 DEFAULT PAR=SPEC
300 IF'SAVE'='Y': PROFILE CSW
400 _END
```

and

```
PROCDEF SYN
100 PARAM TERM, STRING, CSW, SAVE
200 SYNONYM TERM=STRING
300 IF'SAVE'='Y' ; PROFILE CSW
400 _END
```

The user can now issue SYN or DEF; however, if he wants to retain the synonym or default value, he enters SAVE='Y' as an operand of these two commands. When the save option is selected, the user can also enter CSW='Y' if he wants to retain command symbols.

4. The user wants to define a procedure to facilitate editing his message file.

```
procdef msgedit
100 param msgid
200 edit sysulib,sysmlf
300 region msgid
400 _end
```

Since the EDIT command is included in this procedure, with a data set name of SYSMLF, the user always gains access to the message file by issuing MSGEDIT. The only operand required is the value of MSGID (i.e., the message identification) with which the message is associated.

5. The user wants to eliminate the PARAM line from the PROCDEF command.

```
PROCDEF PRCDEF
100 PARAM PNAME, DPAR1, DPAR2, DPAR3, DPAR4
200 DEFAULT SYSINX=E
300 PROCDEF PNAME
400 PARAM DPAR1, DPAR2, DPAR3, DPAR4
500 _DEFAULT SYSINX=G
600 INSERT
700 _END
```

The user can then issue a PROCDEF and PARAM on one line.

```
prcdef        pname, dpar1, dpar2, dpar3, dpar4
```

and then enter the lines that constitute the body of the procedure.

6. The user wants to generate a command to eliminate previously defined procedures; he uses PRCDEF defined in Example 5.

```
PRCDEF EXPUNGE, PNAME
200 EDIT SYSULIB,SYSPRO
300 REGION PNAME
400 _ _EXCISE 0, LAST
500 _END
```

To eliminate the PROCDEF-defined command called OUTPUT that was defined in Example 1, the user issues

```
expunge       output
```

and the procedure shown in Example 1 is eliminated.

7. The command system is provided with a procedure called ZLOGON, which is automatically invoked when a user logs-on. Each user may define the actions he wants performed by ZLOGON with either PROCDEF, BUILTIN, or SYNONYM. For example, a user who uses only one program might create this procedure:

```
procdef zlogon
100 qualify mname=pyroll
200 pyroll
300 _end
```

Now, every time he initiates a task, the system qualifies all internal symbols implicitly, loads his program, and causes the program to start execution.

User Prompter

The user prompter is a message-handling facility that allows the system or the users to locate and display messages, responses, or explanations from one of two message files via the PRMPT macro instruction.

The user can augment these capabilities through the facilities of the command system.

1. Request, via the EXPLAIN command, explanations of entire messages or words from the system.

2. Specify the appropriate filter option in his user profile to indicate the classification of messages he wants to receive during execution of a task.

3. Utilize the text editor to construct a user-message file that will generate specific user-written messages in lieu of system-issued messages.

4. The assembler user may utilize the PRMPT and MCAST macro instructions within his coding to communicate directly with the user prompter. (For detailed descriptions of these macros, refer to Assembler User Macro Instructions.)

Message Files

The system message file is a VISAM member of the VPAM data set SYSLIB and is accessible only to system programmers. The user-message file is a VISAM member of the VPAM data set USERLIB. A user may construct a message with the same identification code as for a system message, but with different text, and then put the new message into the user's message file. For a specific message-identification code, the system searches USERLIB first and then SYSLIB. This allows each user to have his own messages overlaying system messages.

Message Generation

A message is generated to the user by a call on the user prompter with the PRMPT macro instruction that is issued by either the system or the assembler user. The message-identification code in the operand of the PRMPT macro instruction points to a specific message in the message file. The prompter will then display the message on SYSOUT when the PRMPT macro instruction is executed.

EXPLAIN Command

This command allows the user to obtain explanations of entire messages, or of designated words within a message, that the system has generated.

Operation	Operand
EXPLAIN	[{word ORIGIN TEXT,message identification RESPONSE}]

Note: Keyword operand format is invalid.

word

specifies that a word prefixed with an underline, within the text of the last message, is to be explained.

Specified as: a normal or quoted string.

ORIGIN

specifies that the user wants to have displayed the location of the program (the system's or the user's) that caused the message to be generated. Every message has an identification code, which is also displayed.

Specified as: ORIGIN

Note: This form of EXPLAIN assists the user in isolating the module that caused the message generation, and is intended primarily for systems programmers.

TEXT, message identification

specifies that any code-identified message with an underscore preceding its first character position can be displayed in full text.

Specified as: TEXT followed by a comma and the message identification.

Note: The TEXT option is not required for an explanation of the most recently issued message.

RESPONSE

specifies that the possible responses to the last message are to be displayed.

Specified as: RESPONSE

Functional Description: If no operand is used with the EXPLAIN command, the preceding message, if explainable (and that is indicated by an underscore in the character position preceding the message text), is restated more explicitly. If the message is not explainable but contains explainable words (identified by an underscore in the first character position preceding the word) these words are elaborated; otherwise, the system's reply is "no explanation." Two underscores before the first word of an explainable message indicate that the first word can be explained individually.

Programming Notes: The user may use any one, but only one, of the five optional ways of presenting EXPLAIN in one use of the command. If the user wants to use more than one option, he must give additional EXPLAIN commands, in succession, each one containing the single option desired.

Except for the "TEXT,message identification code" form of EXPLAIN, this command must immediately follow the message that is to be explained.

Example: A user is executing a module named UPTCM as a part of his conversational task. This module prompts for the data set organization and this message is displayed at the user's terminal:

```
UPTCM170 _ENTER_VAM DS. ORG.
```

The user does not understand what is required. Since the message text is prefixed with an underscore, the message has an explanation, which the user requests:

```
explain
```

The explanation for the current message UPTCM170 is displayed:

```
UPTCM170 A VAM DATA SET'S ORGANIZATION DEFINES THE
OVERALL RELATIONSHIPS OF THE LOGICAL RECORDS MAKING
UP THE DATA SET.ENTER...VP...OR...VS...OR...VI...
```

After reading the message explanation, the user wants more information about the explainable word, VAM, (it is preceded by an underscore):

```
explain vam
```

A definition of VAM is displayed at his terminal. An explanation message can, in turn, contain explainable words for which further clarification can be requested. Word explanations can continue to any number of levels.

The user eventually understands the message, but now he is uncertain of the form he should use in a valid response; he enters:

```
explain response
```

Now, all possible responses to the message identified by UPTCM170 are displayed:

```
VALID RESPONSES ARE:  VP, VI, VS
```

Later in his terminal session, the user again needs the explanation of UPTCM170; he enters:

```
explain text, uptcm170
```

The TEXT option is necessary here since the explanation requested is not for the most-recently issued message.

Message Filtering

When messages are defined, they are classified as to type, and the user can specify in his user profile, which types he wants displayed. Messages are classified by severity, length, and mode of user's task (conversational or nonconversational, or both). A three-byte field, one byte for each classification, contains the classification codes for each message in the message files. Table 16 shows the message types for each of the three classifications and their corresponding codes. The default codes, established by the system, are underlined.

Severity levels are arranged in increasing order. The user prompter will filter (i.e., block off) messages with a lower severity code (i.e., less severe) than the severity filter code. For example, if the filter code is N, all N, X, and T messages that are requested for normal errors will be displayed; I and W messages will not be displayed.

LIMEN is the default name in the user profile for severity codes. To change the system default for severity, the user issues a DEFAULT command, specifying LIMEN and the desired default value in the operand. For a description of DEFAULT, refer to Part V "User Profile Management."

BREVITY, message length, classifies four different lengths of messages. The shortest message is the message-identification code. The standard message is a short message; the extended message is an amplified version of the standard message. A reference message points to another message that contains the length code. The user prompter will display a message with the same length code as that specified in the user profile.

To alter the system default for message length, the user should specify BREVITY with the desired default value as the operand of a DEFAULT command.

Filtering, based on user's mode of operation, cannot be modified by the user. C messages are displayed only for conversational tasks; B messages only for nonconversational tasks; and A messages for all tasks, regardless of mode.

Table 16. Filter Codes

Severity of Message (LIMEN)		Length of Message (BREVITY)		Mode of Task	
Type	Code	Type	Code	Type	Code
Information	I	Message ID	M	Conversational	C
Warning	W	Standard	S	Nonconversational	B
Normal error	N	Extended	E	All	A
Serious error	X	Reference	R		
Terminate error	T				

Notes: All severity codes and length-code S and E are applicable for both message-classification and filtering options; only length codes M, E and S are applicable as filtering options; messages with length code R have their length codes in the referenced message; mode codes are applicable only for message classification.

Message File Construction

The user message file is a member of the user library (USERLIB) called SYSMLF. The user message file is constructed and maintained by using the facilities of the text editor. New messages that the user prompter can access must be constructed by the text editor in SYSMLF; to initiate this process:

```
edit sysulib,sysmlf
```

The text editor constructs VISAM records from its input commands and data lines. The message-identification code comes from the region specification. The input line number is right-justified in a seven-byte unsigned decimal field and padded with leading 0s. The line text is unchanged from the input data line.

When reference line numbers are entered, their format must match the internal format of the referenced original line number. For example, an original line number entered as 200 will be stored internally in bytes 12-18 of a VISAM record as 0000200. If this line is referenced by another message, the reference field, bytes 12-18 of the line, must contain 0000200.

Many messages require that operands (in most cases, user-defined names) be inserted in specified positions in the message. These positions are indicated in the body of the message by the elements $N, where N can assume the integer values 1 through 20, and denotes the Nth element of the parameter sublist in the calling sequence. The text of explainable-message records, when output, begins with an underscore. Similarly, explainable words in a message body are preceded by the underscore character. All explainable words and messages must have corresponding explanation messages. This example illustrates the procedure for entering a message in the user message file:

```
edit sysulib,sysmlf
region czatpl01
100 _update
0 isa name unknown
end
```

The text editor was invoked with the EDIT command. The REGION command specified the message identification code and the system provided the next available line number (100). The user entered an UPDATE command preceded by an underscore, requesting the system to act on this command immediately. He then entered line 0. The END command signals the text editor to discontinue entry in the message file. ISA is the message classification code.

The user prompter has a reference capability that allows the system and the user to reduce the number of times and identical message might appear in the message file; yet, maintain unique message identification codes for every distinct call to a message. For example, if system message ABC has a desired text and a user message XYZ is established with identical text, message XYZ can point to ABC, rather than repeat the text of ABC. Message XYZ, a reference message, has an R in the second position of the classification field.

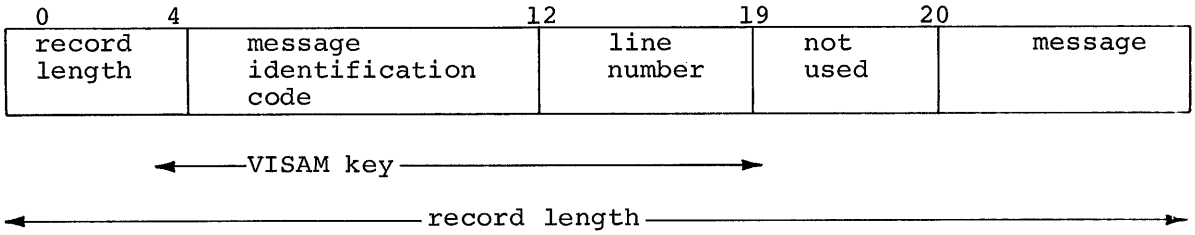
By calling identical messages with more than one message-identification code, reference messages permit a message's identification to convey information as to its source.

Message Types and Format

There are five types of messages in the system message file.

Standard message	-	a brief communication from an object module to the user; begins at line 0.
Extended message	-	a lengthier, more detailed communication from an object module to the user, begins at line 100
Response	-	a set of words, each with a unique code identifying the response expected from the user; it begins at line 200.
Explanation message	-	an explanation of the standard message; it begins at line 300.
Word explanation	-	an explanation of an explainable word in a message; it begins at any line above 399.

The VISAM variable length record format is:



The line format varies with the type of line (see Table 17).

Table 17. Message-Line Format

Message Types	Message Classification Bytes 20 - 22	Byte 23	Message Text Bytes 24-end of line	
Standard Message	3-byte classification code	Underscore for explainable message	Message Text Reference MSG ID, bytes 24 - 31	Reference-line number, bytes 32-38
Extended Message	3-byte classification code	Underscore for explainable message	Message Text Reference Msg ID, bytes 24 - 31	Reference-line number, bytes 32-38
Response	3-byte classification code	Not used	Message Text word 1=code 1, word 2=code 2,etc Reference msg ID, bytes 24 - 31	Reference-line number, bytes 32-38
Message Explanation	3-byte classification code	Not used	Message Explanation Text Reference msg ID, bytes 24 - 31	Reference-line number, bytes 32-38
Word Explanation	3-byte classification code	Not used	Explain word, bytes 24-31	Word Explanation Text Reference scope, bytes 32-39 (not used) / Reference-line number bytes, 40-46; reference word, bytes 48-55

Word Explanation Scope

The scope of word explanations vary, depending on the word. For example, some word explanations will be universal in scope. Other word explanations might be universal within a major component of TSS/360, such as the Command System. Still others will have a scope limited to the particular message in which they appear. Broad scopes are an advantage because fewer explanation records are needed and the size of the message file can be minimized. The user prompter enables the user to regulate the scopes of word explanations to favor his particular operation.

The scope of a word explanation is indicated by its eight-byte message identification code. Explanations with universal scope have an all-blank identification code. For scopes restricted to a single message, the full eight-byte message-identification code is used. Identification codes can be assigned in a pattern to allow various levels of scope, between universal and fully restrictive. For example, assume the command system divides into three main modules: CZASA, CZATE, and CZATP. All command-system-message identification codes start with CZA, followed by a three-digit sequential message number within the module. As an example, five messages in module CZATP could be identified:

```
CZATP101
CZATP102
CZATP103
CZATP104
CZATP105
```

Words whose meaning is universal in scope within the command system would have an identification code of CZA. If the scope were limited to a particular module, say CZATP, the identification code would be CZATP. The scope is further restricted to a particular message by adding the final three digits.

The user prompter provides two one-byte masks, one for the system-message file and one for the user-message file, to allow users to control the user prompter's search for word explanations. Each bit position in the mask corresponds to a byte in the identification code of the message that contains the explainable word. The correspondence is from left to right, positions 0 through 7, respectively. Bits are turned on to indicate how many bytes of the current message-identification code are to be compared in searching for a word-explanation record. The user prompter scans the mask, starting on the right with bit 7, looking for 1-bits. Each 1-bit causes an access to the message file, using as a search argument the current message-identification bytes, from position 0 through the position corresponding to the mask bit.

Both system and user masks are located in the profile character and switch table, a section of the user profile (see Appendix C). The values may be changed by using the MCAST macro instruction (refer to Assembler User Macro Instructions). The system and user masks will be initially set as a part of the prototype profile, but may be changed by a user.

APPENDIX A: BULK INPUT FROM MAGNETIC TAPE

The user can enter his data sets that are on tape into the system. The way described here is the only direct means of reading a data set from tape, and then converting to VAM organization, writing onto public storage, and cataloging. The user must send information, indicated below, with his tape, to the system operator. He must also ensure that his tape format meets system requirements, defined later in this appendix. The data set that will be stored and cataloged has a different organization from that residing on the tape (i.e., the input data set) and must, therefore, have its own data set name. When the new data set has been cataloged, the user can refer to it just as he would to any other cataloged data set belonging to him.

Information Needed by System Operator

The user must send this information with his tape to the system operator, for every data set that is to be read and cataloged:

Operands
{volume identification,[tape type] / CTLG}, user identification, input data set name, cataloged data set name,[LINE],[error]

volume identification

volume identification of the user's tape

Specified as: one-to-six alphameric characters

tape type

type of tape

Specified as: 7 - seven-track
7DC - seven-track with data converter feature
9 - nine-track

System default: tape type specified at system generation time is assumed

CTLG

input data set has previously been cataloged

Specified as: CTLG

user identification

identification of user to whom the data set belongs

Specified as: three-to-eight alphameric characters; the first must be alphabetic

input data set name

name of the input data set

Specified as: fully qualified data set name

cataloged data set name

name under which the data set is to be cataloged

Specified as: a fully qualified data set name

LINE

lines are to be numbered. Each logical record in the VISAM data set that is created will be prefixed by a seven-character line number of the form xxxxx00, and by a byte of binary 0s that is reserved for system use. The result will be a line data set with variable format records.

Specified as: LINE
System default: no line numbering; organization, VSAM

error

specifies the action to be taken if an uncorrectable read-error occurs; one of three options may be selected.

Specified as: ACCEPT - error record will be accepted
SKIP - error record will be skipped
END - read operation will be terminated
System default: END is assumed

Functional Description: The system will read the input data set, convert it to VAM organization, store it on public storage, and catalog it in the user's catalog under the name specified in the cataloged data set name operand. If the input tape contains more than one data set, the system will read the specified input data set only.

The data set that is stored on public storage has either VSAM or VISAM organization, depending on whether the LINE option was selected. If line numbering was requested, the system will generate line numbers using an increment value of 100. The maximum number of logical records permitted, therefore, is 100,000. The input data set record length must not exceed 120 bytes if line numbering is requested.

The system does not perform code conversions. However, if the data set is on seven-track tape, the system will make any character adjustments required for data validity.

Programming Notes: When the user wants to submit a data set on seven-track tape (with or without data converter feature), he must first consider tape characteristics. He merely enters the tape-type operand, as shown in the preceding format, if characteristics such as density and parity, match the standards set by the installation. But, if he wants other characteristics, he must:

1. Issue a DDEF command for the data set, specifying the tape characteristics,

2. Issue a CATALOG command to catalog the data set.

3. Specify the CTLG operand before sending the tape to be read.

The system operator will use the information supplied by the user to enter an RT command that causes execution of a system-provided task to handle the tape input. The SYSOUT listing of that task, which is returned to the user, may contain messages.

Tape Format Requirements

The magnetic tape must have the standard TSS/360 label. Physical records must be of fixed length and no longer than 32,767 bytes.

APPENDIX B: BULK INPUT FROM CARD DECKS

The user can submit his data sets on punched cards to the system operator, who will then enter them into the system via a high-speed card reader. Two types of input data sets are permitted: nonconversational SYSIN data sets and data-card data sets. The two types may be interspersed, one following another, in any order within a batch of cards. The rules for setting up these data sets are given below.

Note: When the user wants to enter a nonconversational SYSIN data set together with the data sets it references, he must be certain that the data sets precede the SYSIN data set. The system, generally, will try to execute the SYSIN data set as soon as it has been read.

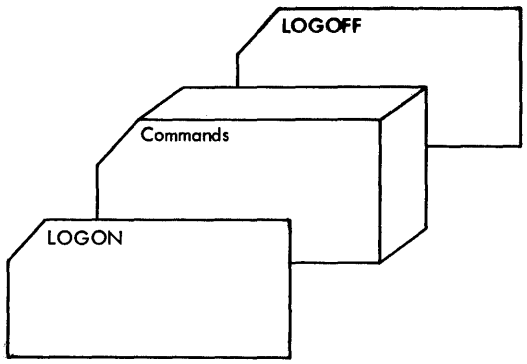
The acceptable character set for punched cards is described in Terminal User's Guide.

Nonconversational SYSIN Data Set

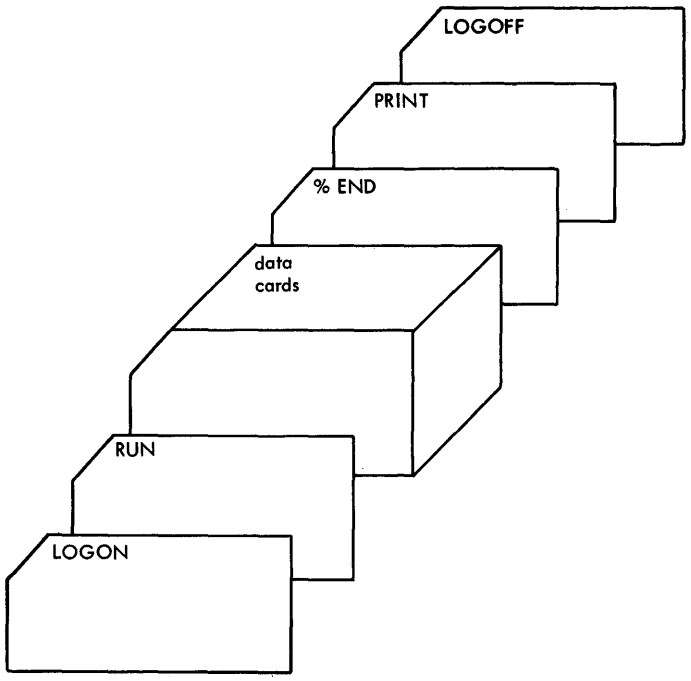
A nonconversational data set contains all commands needed to run a nonconversational task. These commands are punched, each starting in column 3 of a new card, in exactly the format used to enter commands from a terminal (see Part I, "Command Format and Notation"). The first card must be a LOGON command; the last, LOGOFF. Other commands are as required for the particular task.

When the data set is read in, it becomes the SYSIN data set of a nonconversational task; it is executed as soon as space is available. After execution, the SYSIN data set is eliminated; it does not remain in the catalog or in system storage.

The card-deck format is:



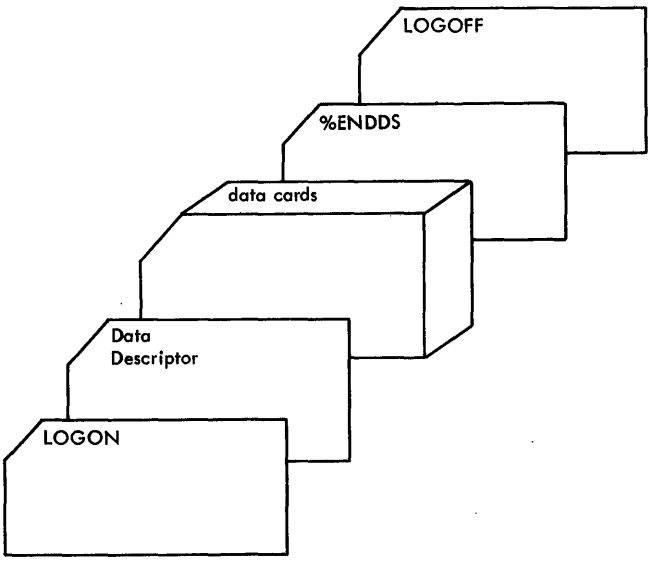
The SYSIN data set may include data that is to be read by the user's object program at execution time. If so, the data to be read must appear immediately after the command that will start execution of the user's program. (Unlike commands, card data may start in column 1.) Also, the end-of-data card, with %END starting in column 3, must follow the last data card, as in this example:



Data-Card Data Set

This type of data set contains any information the user wants to put into public storage as a cataloged data set; it may include commands. When this type of data set is read, a VAM data set is created and cataloged in public storage. This VAM data set will continue to reside in storage until it is specifically erased. Unlike the nonconversational SYSIN data set, it is not executed upon being read.

The first card of the data set must be LOGON; the last, LOGOFF. Two control cards are also mandatory: a data descriptor card, immediately after LOGON, and %ENDDS card, just before LOGOFF, as in the card deck format:



Data Descriptor Card

This card, which provides information needed by the system to create the desired data set, is in this format:

Col.
3

Operands
DATASET, data set name,[format],[starting number],[ending number], [LINE],[error]

DATASET

indicates that data descriptor information follows

Specified as: DATASET

data set name

name under which the new data set is to be cataloged

Specified as: a fully qualified data set name

format

the class of card punching to be used

Specified as: EBCDIC - extended binary coded decimal interchange code
BCD - binary coded decimal
System default: EBCDIC is assumed

starting number

the first card column to be read when creating a data set record

Specified as: decimal number from 1 to 79
System default: column 1 is assumed

ending number

last column to be read when creating the data set records

Specified as: decimal number; maximum, 80

LINE

indicates that line numbering is requested. Each record in the data set will be prefixed by a seven-character line number of the form XXXXX00, and by a byte of binary 0s that is reserved for system use. The resulting data set will be a line data set with variable format records.

Specified as: LINE
System default: no line-numbering organization, VSAM

error

indicates the action to be taken if an uncorrectable read-error occurs

Specified as: ACCEPT - accept the record in error
SKIP - skip the entire logical record if any card in it is in error
END - terminate reading of the data set
System default: END is assumed

Functional Description: The operator issues an RC command and the system will read the input data set, convert it to VAM organization, put it in public storage, and catalog it in the user's catalog under the data set name provided. The stored data set has either VSAM or VISAM organization, depending on whether the line option was selected. If line numbering was requested, the system will generate line numbers, using an increment value of 100. The maximum number of lines permitted in such a data set is, therefore, 100,000. When line numbering is requested, the new data set record length must not exceed 120 bytes.

Cautions: Data cards may be punched beginning in column 1. LOGON and LOGOFF, beginning in column 3, delimit successive input jobs. Therefore, they should not begin in column 3 when used in a data set created by the DATA command.

%ENDDS Card

This card, with %ENDDS starting in column 3, marks the end of a data set that is to be cataloged.

Note: Submitting cards for bulk input results in a system-provided task being executed to handle the card input. The SYSOUT listing of that system task may contain messages.

APPENDIX C: PROTOTYPE PROFILE

Listed below are the IBM system-supplied values contained in the prototype profile. These values are principally the keywords and values used by the system and the initial character-translation table used by GATE.

COMMAND SYSTEM DEFAULTS

OPERAND NAME	DEFAULT VALUE	PURPOSE OR USERS
ACC		CATALOG
ACCESS		PERMIT
ACTION	O	CATALOG
ASMLIST	Y	ASM
AUTH	U	JOIN
BASE		CDS, DATA, NUMBER
BCD	N	FTN
BREVITY	S	Message Length Filter
BSN		CANCEL
CBIN	E	PUNCH
CHARGE		JOIN, LOGON
CONF		MODIFY
COUNT	30000	Number of Lines for Bulk Output
CRLIST	N	ASM, FTN
CSW	N	PROFILE
CTLG		RT
DCB		DDEF
DDNAME		DDEF, EDIT, EXCERPT, RELEASE
DISP		CATALOG, CDS, DDEF
DSNAME		BACK, CATALOG, CDD, DATA, DDEF, DELETE, DSS?, ERASE, EXECUTE, LINE?, MODIFY, PERMIT, POD?, PRINT, PUNCH, SHARE, WT
DSNAME1		CDS, RT, TV, VT, VV
DSNAME2		CDS, RT, TV, VT, VV, WT
DSORG	VI	DDEF
EMPTY		RC, SHUTDOWN
ENDNO		PRINT, PUNCH, WT
ERASE		PRINT, PUNCH, WT
ERROROPT	END	PRINT, PUNCH, WT
EXTNAME		BUILTIN
FACTOR		WT
FORM		PRINT, PUNCH
GDG		CATALOG
GNO		CATALOG
HEADER		PRINT, WT
INCR		CDS, DATA, INSERT, NUMBER, REVISE,
INSTLOC		BRANCH
ISD	Y	ASM, FTN, LNK
ISDLIST	N	ASM
KEYLEN		MODIFY
LABEL		MODIFY
LIB		LNK

COMMAND SYSTEM DEFAULTS (Cont'd.)

OPERAND NAME	DEFAULT VALUE	PURPOSE OR USERS
LIMEN	W	Message Severity-Filter
LINCR	(100,100)	ASM, FTN
LINE		LINE?, RT
LOC		RUN
LRECL		MODIFY
MACROLIB		ASM
MINS		TIME
MMAP	N	FTN
MNAME		EDIT, QUALIFY
MSGNO		REPLY
N1		CONTEXT, CORRECT, EXCERPT, EXCISE, INSERT, LIST, LOCATE, NUMBER, REVISE
N2		CONTEXT, CORRECT, EXCERPT, EXCISE, LIST, LOCATE, NUMBER, REVISE
NAME		ASM, BUILTIN, CALL, FTN, LNK, LOAD, PROCDEF, UNLOAD
NAMES		DSS?
NEWNAME		CATALOG
OBLIST	N	FTN
OPTION		DDEF
OWNERDS	*ALL	SHARE
PAGE		PRINT, WT
PASSWORD		JOIN
PMDLIST	N	ASM, FTN, LNK
PREXPAND	X	Controls Procedure Expansion Error Analysis
PRIORITY	9	JOIN
PRIV	D	JOIN
PRTSP	1	PRINT, WT
PUBLIC	N	FTN
RECFM	V	MODIFY
RKP	0	MODIFY
RNAME		EXCERPT, REGION
RSVP		Controls Responses in GATE
RTYPE		DATA
SCOL	0	CORRECT
SETNAME		MODIFY
SLIST	Y	FTN
SOURCE		EDIT
SPACE		DDEF
STACK	1	PUNCH
STARTNO		PRINT, PUNCH, WT
STATE	N	CATALOG, PERMIT
STEDIT	N	ASM, FTN
STORED	N	ASM, FTN, LNK
STRING		LOCATE
STRING1		CONTEXT
STRING2		CONTEXT
SYMBOL		RUN

COMMAND SYSTEM DEFAULTS (Cont'd.)

OPERAND NAME	DEFAULT VALUE	PURPOSE OR USERS
SYMLIST	N	ASM
SYSIN	K	Controls GATE's Access to SYSIN Data Set
SYSINX	G	Controls Command Analyzer's Access to GATE
TEXT		BCST, MSG, REPLY
UNIT		DDEF
USERID		FORCE, JOIN, MSG, QUIT, PERMIT, RT, SHARE
VERID		ASM, FTN, LNK
VOLUME		DDEF, RT, WT
*$@%	*$@%	CORRECT

Character and Switch Table, Primary Dictionary

The user profile contains two additional tables, the Profile Character and Switch Table and the Primary Dictionary. The primary dictionary contains commands written by both the system and the user. Command procedures were discussed in Part VI. The profile character and switch table is made up of two parts, a character-translation table and a table of miscellaneous control characters.

The character-translation table is a list of all characters recognized by the system, each with its internal code and a function code. Function codes available are:

1. Code 00 - Translate only. When this code is specified for a character, the system picks up the internal code of that character.

2. Code 04 - Character kill. Every time it encounters a character with this function code, the system deletes that character and the one preceding it. The system-supplied value is backspace.

3. Code 08 - End-of-block or new line. When the system encounters a character with this code, it recognizes it as the end of an input stream and appends at that point an EOB character from the miscellaneous control-character table. Any characters beyond the EOB character are ignored.

4. Code 0C - Cancel. If the last character in the line has this code the system deletes the character and the entire line that precedes it. The system supplied value is #.

5. Code 10 - Terminal null. If this code is assigned to the last character before the end-of-block, the system will ignore the character. The system supplied value is a new line.

6. Code 14 - Null. Any character to which this code is assigned will be ignored as input to the system.

The user who wants to modify this table must build his own character-translation table in memory, assigning function codes to whatever character he wants. For example, he might assign code 14 (null) to all nonprinting characters. He would then use an MCAST macro instruction in his object program; MCAST is described in Assembler Users Macro Instructions.

PROTOTYPE CHARACTER TRANSLATION TABLE

Translation Character Byte Position	Code	Character	Function Character Byte Position	Code
0	0		256	0
1	1		257	0
2	2		258	0
3	3		259	0
4	4	PF	260	0
5	5	HT	261	0
6	6	LC	262	0
7	7	DEL	263	0
8	8		264	0
9	9		265	0
10	A		266	0
11	B		267	0
12	C		268	0
13	D		269	0
14	E		270	0
15	F		271	10
16	10		272	4
17	11		273	0
18	12		274	0
19	13		275	0
20	14	RES	276	0
21	15	NL	277	0
22	16	BS	278	0
23	17	IL	279	0
24	18		280	0
25	19		281	0
26	1A		282	0
27	1B		283	0
28	1C		284	0
29	1D		285	0
30	1E		286	0
31	1F		287	0
32	20		288	0
33	21		289	0
34	22		290	0
35	23		291	0
36	24	BYP	292	0
37	25	LF	293	0
38	26	EOB	294	8
39	27	PRE	295	0
40	28		296	0
41	29		297	0
42	2A	SM	298	0
43	2B		299	0
44	2C		300	0
45	2D		301	0
46	2E		302	0
47	2F		303	0
48	30		304	0
49	31		305	0
50	32		306	0
51	33		307	0
52	34	PN	308	0
53	35	RS	309	0
54	36	UC	310	0
55	37	EOT	311	0

PROTOTYPE CHARACTER TRANSLATION TABLE (Cont'd.)

Translation Character Byte Position	Code	Character	Function Character Byte Position	Code
56	38		312	0
57	39		313	0
58	3A		314	0
59	3B		315	0
60	3C		316	0
61	3D		317	0
62	3E		318	0
63	3F		319	0
64	40		320	0
65	41		321	0
66	42		322	0
67	43		323	0
68	44		324	0
69	45		325	0
70	46		326	0
71	47		327	0
72	48		328	0
73	49		329	0
74	4A		330	8
75	4B		331	0
76	4C		332	0
77	4D		333	0
78	4E		334	0
79	4F		335	0
80	50	&	336	0
81	51		337	0
82	52		338	0
83	53		339	0
84	54		340	0
85	55		341	0
86	56		342	0
87	57		343	0
88	58		344	0
89	59		345	0
90	5A	!	346	0
91	5B	$	347	0
92	5C	*	348	0
93	5D	)	349	0
94	5E	;	350	0
95	5F		351	0
96	60	-	352	0
97	61	/	353	0
98	62		354	0
99	63		355	0
100	64		356	0
101	65		357	0
102	66		358	0
103	67		359	0
104	68		360	0
105	69		361	0
106	6A		362	0
107	6B	,	363	0
108	6C	%	364	0
109	6D	_	365	0
110	6E		366	0

PROTOTYPE CHARACTER TRANSLATION TABLE (Cont'd.)

Translation Character Byte Position	Code	Character	Function Character Byte Position	Code
111	6F	?	367	0
112	70		368	0
113	71		369	0
114	72		370	0
115	73		371	0
116	74		372	0
117	75		373	0
118	76		374	0
119	77		375	0
120	78		376	0
121	79		377	0
122	7A	:	378	0
123	7B	#	379	C
124	7C	@	380	0
125	7D	'	381	0
126	7E	=	382	0
127	7F	"	383	0
128	80		384	0
129	81	a	385	0
130	82	b	386	0
131	83	c	387	0
132	84	d	388	0
133	85	e	389	0
134	86	f	390	0
135	87	g	391	0
136	88	h	392	0
137	89	i	393	0
138	8A		394	0
139	8B		395	0
140	8C		396	0
141	8D		397	0
142	8E		398	0
143	8F		399	0
144	90		400	0
145	91	j	401	0
146	92	k	402	0
147	93	l	403	0
148	94	m	404	0
149	95	n	405	0
150	96	o	406	0
151	97	p	407	0
152	98	q	408	0
153	99	r	409	0
154	9A		410	0
155	9B		411	0
156	9C		412	0
157	9D		413	0
158	9E		414	0
159	9F		415	0
160	A0		416	0
161	A1		417	0
162	A2	s	418	0
163	A3	t	419	0
164	A4	u	420	0
165	A5	v	421	0

PROTOTYPE CHARACTER TRANSLATION TABLE (Cont'd.)

Translation Character Byte Position	Code	Character	Function Character Byte Position	Code
166	A6	w	422	0
167	A7	x	423	0
168	A8	y	424	0
196	A9	z	425	0
170	AA		426	0
171	AB		427	0
172	AC		428	0
173	AD		429	0
174	AE		430	0
175	AF		431	0
176	B0		432	0
177	B1		433	0
178	B2		434	0
179	B3		435	0
180	B4		436	0
181	B5		437	0
182	B6		438	0
183	B7		439	0
184	B8		440	0
185	B9		441	0
186	BA		442	0
187	BB		443	0
188	BC		444	0
189	BD		445	0
190	BE		446	0
191	BF		447	0
192	C0		448	0
193	C1	A	449	0
194	C2	B	450	0
195	C3	C	451	0
196	C4	D	452	0
197	C5	E	453	0
198	C6	F	454	0
199	C7	G	455	0
200	C8	H	456	0
201	C9	I	457	0
202	CA		458	0
203	CB		459	0
204	CC		460	0
205	CD		461	0
206	CE		462	0
207	CF		463	0
208	D0		464	0
209	D1	J	465	0
210	D2	K	466	0
211	D3	L	467	0
212	D4	M	468	0
213	D5	N	469	0
214	D6	O	470	0
215	D7	P	471	0
216	D8	Q	472	0
217	D9	R	473	0
218	DA		474	0
219	DB		475	0
220	DC		476	0
221	DD		477	0
222	DE		478	0

PROTOTYPE CHARACTER TRANSLATION TABLE (Cont'd.)

Translation Character			Function Character	
Byte Position	Code	Character	Byte Position	Code
223	DF		479	0
224	E0		480	0
225	E1		481	0
226	E2	S	482	0
227	E3	T	483	0
228	E4	U	484	0
229	E5	V	485	0
230	E6	W	486	0
231	E7	X	487	0
232	E8	Y	488	0
233	E9	Z	489	0
234	EA		490	0
235	EB		491	0
236	EC		492	0
237	ED		493	0
238	EE		494	0
239	EF		495	0
240	F0	0	496	0
241	F1	1	497	0
242	F2	2	498	0
243	F3	3	499	0
244	F4	4	500	0
245	F5	5	501	0
246	F6	6	502	0
247	F7	7	503	0
248	F8	8	504	0
249	F9	9	505	0
250	FA		506	0
251	FB		507	0
252	FC		508	0
253	FD		509	0
254	FE		510	0
256	FF		511	0
			512	0

The table of miscellaneous control characters includes:

1. Source list EOB character; defines the end of an input block. Its initial value is X'26'.

2. Command system continuation character. Normally, an EOB occurs when the carriage is returned. If the last character before a carriage return is a command system continuation character, the line input is continued past the carriage return. The intial representation of the default for this character is a hyphen (X'60).

3. Command system prefix character. Entry of this character requires the system to execute immediately the command following the character. Initially this character is defined as an underscore (X'6D').

4. Transient statement prefix character. This character is included for future use with advanced language processors. When the processor encounters the character, it ceases processing the input string as data and sends whatever follows to a predetermined entry point for execution. The system initially defines this character as a vertical stroke (X'4F').

5. Preferred line-break indicator. With varying line-length capabilities of different devices, it may become necessary to divide a line of input. A user can enter this character to indicate when he would prefer to have a line broken if it becomes necessary to do so. For ordinary printed text he might make the character a space. The initial character is X'72'.

6. System scope mask. This character controls searches for explanatory messages issued by the user prompter. Its default value is X'29' (00101001). Using this mask, a search for a particular explanatory message would begin with a search of the messages that have eight-character identifications. If none is found, messages with five-character identifications would be searched, then three-character identifications. As the identifications get shorter, the messages become more general in scope. The system scope mask is used only with the system-messages file on SYSLIB; the mask may not be changed by the user.

7. User scope mask. This character works like the system scope mask, but on user-created messages in the USERLIB. The user may set this mask according to his own search logic. The default value is 00101001.

8. Command prompter. This may be a string of up to eight characters. Initial default is an underscore followed by a backspace. The system uses this string to prompt for a command.

9. SYSIN keyboard/card reader switch. This switch indicates the type of device from which input will be accepted by the system. It may be set with a K for a terminal keyboard, or with an E to indicate either the keyboard or the card reader. If a user specifies K as the switch setting, the system will not recognize his subsequent specification of a card reader as the input device. Initial value is E.

10. Carriage-return suppression character. When this character is the last in a message being written to SYSOUT by the command system, the character is suppressed and the system does not add a new line character to the text. The system-supplied value is colon (X'7A').

MACHINE CODES

Function	Byte Value (hexadecimal)
Write (no automatic space)	01
Write and space 1 line after printing	09
Write and space 2 lines after printing	11
Write and space 3 lines after printing	19
Write and skip to channel 1 after printing	89
Write and skip to channel 2 after printing	91
Write and skip to channel 3 after printing	99
Write and skip to channel 4 after printing	A1
Write and skip to channel 5 after printing	A9
Write and skip to channel 6 after printing	B1
Write and skip to channel 7 after printing	B9
Write and skip to channel 8 after printing	C1
Write and skip to channel 9 after printing	C9
Write and skip to channel 10 after printing	D1
Write and skip to channel 11 after printing	D9
Write and skip to channel 12 after printing	E1

Note: To obtain the corresponding carriage-control operations (space or skip to channel N) without printing, increase the value of the low-order digit by hexadecimal 2. Example:

space two lines	13
skip to channel 5	AB
skip to channel 9	CB

EXTENDED USASI CODES

Function	Character
Skip no line before printing	+
Skip 1 line before printing	blank
Skip 2 lines before printing	0

EXTENDED USASI CODES (cont'd)

Function	Character
Skip 3 lines before printing	-
Skip to channel 1 before printing	1
Skip to channel 2 before printing	2
Skip to channel 3 before printing	3
Skip to channel 4 before printing	4
Skip to channel 5 before printing	5
Skip to channel 6 before printing	6
Skip to channel 7 before printing	7
Skip to channel 8 before printing	8
Skip to channel 9 before printing	9
Skip to channel 10 before printing	A
Skip to channel 11 before printing	B
Skip to channel 12 before printing	C

IBM 2540 PUNCH MACHINE CODES

Function	Data Mode 1 (byte value)	Data Mode 2 (hexadecimal)
TYPE AA		
Read, feed, and select stacker R1	02	22
Read, feed, and select stacker R2	42	62
Read, feed, and select stacker RP3	82	A2
TYPE AB		
Read and no feed or stacker selection	C2	E2
Read, feed, and no stacker selection	D2	F2
TYPE BA		
Feed and select stacker R1	23	23
Feed and select stacker R2	63	63
Feed and select stacker RP3	A3	A3
PFR write, feed, and select stacker P1	09	29
PFR write, feed, and select stacker P2	49	69
PFR write, feed, and select stacker RP3	89	A9
TYPE BB		
Write, feed, and select stacker P1	01	21
Write, feed, and select stacker P2	41	61
Write, feed, and select stacker RP3	81	A1

EXTENDED USASI CODES

Function	Character
Select punch pocket 1	V
Select punch pocket 2	W

APPENDIX F: DETAILED DESCRIPTION OF DDEF COMMAND

This appendix describes the DDEF command as used to define any private, or atypical public, data sets. To define typical public data sets (i.e., those with virtual access method (VAM) sequential organization on direct-access public storage, arranged in pages and having standard labels), see the DDEF command description in Part III.

Table F-1 lists the required and optional fields of the DDEF command for various types of data sets. The complete command format illustration of the DDEF command is:

Operation	Operand
DDEF	DDNAME=data definition name [,DSORG={MS|CS|PS|RX|VI|VP|VS}] ,DSNAME=data set name [,DCB=([data definition name][,DSORG=data set organization] [,MACRF=type of macros][,BUFL=buffer length] [,DEVD=device type][,BUFNO=number of buffers] [,BFTEK=buffer technique] [,NCP=consecutive macro number] [,RECFM=record format][,OPTCD=W][,LRECL=record length] [,BLKSIZE=block length][,KEYLEN=key length] [,PRTSP=spacing][,STACK=stacker selection] [,DEN=tape density] [,MODE=mode of operation][,TRTCH=data conversion] [,EROPT=error option][,PAD=padding] [,RKP=key displacement][,IMSK=error recovery procedures]] [,UNIT=({AFF=data definition name DA[,data type] TA[,tape type]})] [,SPACE=({CYL|TRK|record length},primary[,secondary][,HOLD])] [,VOLUME=([{PRIVATE|volume sequence}][,volume serial number[,...]])] [,LABEL=([file sequence number][,{WL|SL|SUL}] [,RETPD=retention period])] [,DISP={MOD|OLD|NEW}] [,OPTION={CONC|JOBLIB}]

DDNAME

specifies the symbolic data definition name that is associated with the data set, and which provides a link between the data control block (DCB) in the user's program and the data set definition.

Specified as: one-to-eight alphameric characters, the first must be alphabetic; DDNAME may not begin with SYS, since these characters are used to prefix system-reserved data definition names.

Note: When this command is utilized by an assembler language user, the value specified for this operand must be the value specified for the DDNAME operand in the DCB macro instruction. When this command is employed by a FORTRAN user, the value specified for this operand must be "FTNF00xx" where "xx" is the two-digit device address used to reference the data set in his FORTRAN program.

DSORG

indicates the organization of the data set being defined.

Specified as:
- MS - MSAM (multiple sequential access method)
- CX - TAM (terminal access method)
- PS - QSAM or BSAM (physical sequential access methods)
- RX - IOREQ (I/O request)
- VI - VISAM (virtual indexed sequential access method)
- VP - VPAM (virtual partitioned access method)
- VS - VSAM (virtual sequential access method)

System default: VI is assumed

DSNAME

specifies the name under which the data set may be cataloged or referred to for temporary reference.

Specified as: a fully qualified data set name or member name of VPAM data set; when specified, the member name is enclosed in parentheses and immediately follows the VPAM data set name.

Note: When a data set created under System/360 Operating System is introduced to TSS/360 for the first time, the value specified for DSNAME must be preceded by an asterisk (*). Subsequent references to this data set are not prefixed by the asterisk. The data set name preceded by an asterisk may have a maximum of 44 characters.

DCB

specifies data control block information.

Specified as: the data definition name, preceded by an asterisk (*), is the data definition name of a previously issued DDEF command. This indicates that the data control block of the preceding DDEF command is to be duplicated for the current DDEF command.

Detailed descriptions of the DCB suboperands are given in Assembler Programmer's Guide and FORTRAN Programmer's Guide.

Note: If the data set is or will be on tape, the DEN suboperand must be furnished to specify tape density, unless the tape conforms to the DEN default value, which is set at system generation time.

UNIT

specifies the type of device required by the data set. Direct-access devices may be specified for either public or private volumes. The other types of devices and unit affinity may be specified for private volumes only. Allowable kinds of devices are specified at system generation time, and therefore may be changed.

Specified as:

AFF=data definition name -- unit affinity. The data set being defined is to be assigned the same device reserved for the data set identified by a data definition name, which is the definition name of a previously issued DDEF command. This subfield is unacceptable if the data set being defined is new and is to be on a direct-access device.

DA[,datatype] -- a direct-access device is required for the data set. When entered, data type specifies the type of direct-access device being specified, as expressed in a four-digit number.

System default: the type of direct-access device specified at system generation time is assumed.

TA[,tape type] -- a magnetic tape device is required for the data set.

Specified as: 7 -- seven-track tape
7DC -- seven-track tape with data converter feature
9 -- nine-track tape

System default: the type of tape device specified at system generation time is assumed.

SPACE

indicates the direct-access storage allocation for the data set.

Specified as:

TRK - space requirements are expressed in terms of tracks.
CYL - space requirements are expressed in terms of cylinders.
(record length) a decimal number not exceeding 32,767, specifying the average length of the physical records.

System default: If the data set organization is SAM, the space requirements are assumed to be expressed in terms of cylinders. If the data set organization is VAM, the space requirements are assumed to be in pages (of 4096 bytes).

(primary) a one-to-three-digit decimal number specifying the amount of space in terms of tracks or cylinders.

(secondary) a one-to-three-digit decimal number specifying the amount of additional space to be allocated each time the space allocated to the data set, as specified in "primary," has been exhausted and more data is to be written.

System default: the secondary space allocation specified at system generation time is assumed.

HOLD

the unused storage assigned to the data set being defined is not to be released when the data set is closed.

Specified as: HOLD

System default: the unused storage will be released when the data set is closed.

Note: If the SPACE operand is not specified, the direct-access storage allocation specified at system generation time is assigned.

LABEL

specifies the labeling conventions.

Specified as: (file sequence number) a one-or-two-digit decimal number specifying the file sequence number of a data set residing on a tape and that has multiple data sets on a tape volume.

System default: the data set is assumed to be the first (or only) one on the tape volume.

These three suboperands specify either the type of labeling desired or the absence of labeling:

NL - no labels
SL - standard labels (as specified at system generation time)
SUL - standard and user labels

System default: SL is assumed

RETPD = retention period

Specified as: a four-digit decimal number specifying the number of days the data set is to be retained by the system; this suboperand is applicable for data sets on direct-access volumes or labeled tapes.

System default: the retention period is assumed to be 0-days, allowing immediate rewriting.

Note: If the entire LABEL operand is defaulted, the labeling conventions specified at system generation time are assumed, unless the data set being defined is already cataloged, in which case, label information is retrieved from the catalog.

VOLUME

the volume on which the data set resides. Normally, this field is used for an uncataloged data set that resides on a private volume.

Specified as:

PRIVATE - specifies that volumes are to be allocated from the system pool (i.e., the scratch or disk available to the operator). Once assigned, the volume remains the user's, exclusively, until he notifies the operator that it may be returned to the pool.

(volume sequence number) a one-to-four-digit number specifying the first volume of the data set to be read or written. It is meaningful only if the data set has SAM organization, is cataloged, and its earlier volumes are to be skipped.

(volume serial number) one-to-six alphameric characters specifying the volume serial numbers that identify the volumes on which the data set resides. This suboperand is required for old, uncataloged data sets that reside on private volumes; it may be supplied for new data sets that will reside on private volumes.

System default: If volume sequence number is specified, the data set is cataloged and the serial numbers are retrieved from the catalog. If PRIVATE is specified, this suboperand must be omitted and a volume serial number will be assigned by the system.

Note: the entire operand may be defaulted if a new data set is to be created on a public volume or if an old cataloged data set is being defined.

DISP

specifies whether the data set already exists or is to be created.

Specified as:

OLD - data set being defined exists
NEW - data set being defined is new; has not yet been created
MOD - data set being defined exists; an addition to it is being made

Note: MOD, which applies only to SAM data sets on private volumes, causes logical positioning after the last record of the data set.

System default: If the data set is cataloged, OLD is assumed; if not cataloged, NEW is assumed.

Note: If the user specifies DISP as OLD, NEW, or MOD and this does not agree with the actual state of the data set, then:

In conversational mode, the user receives a diagnostic message so that he can correct this error.

In nonconversational mode, the task is abnormally terminated.

OPTION

specifies that either a job library is being defined or a data set is being added to the concatentated data set named in the DDNAME operand.

Specified as:

CONC - Only SAM data sets that are not job libraries can be concatenated and with one or more data sets having the same data definition names. The order of concatenated data sets is the same as the order in which they are defined.

JOBLIB - specifies that the data set being defined is to be used as a job library. The data set name specified in the DSNAME operand is entered into the program library list.

Functional Description: The DDEF command causes a system entry to be established for the data set definition that can be referenced by allocation routines and access methods. The link between this definition and the problem program's reference to the data set (i.e., the data control block) is the data definition name. The entry containing the definition is maintained until the user logs off or until, through the RELEASE command, the data set definition is deleted.

The DDEF command also results in a request, when necessary, for device allocation and volume mounting, when the defined data set is private and resides on a demountable volume as a reel of tape or a disk pack.

Programming Notes: The DDEF command that defines a cataloged data set is brief and simple. The only required operands are DDNAME, DSNAME, and DISP. Other operands are unnecessary since the organization of the data set is described in its catalog entry.

DDEF commands that define uncataloged data sets may be divided into two groups: those defining new data sets (i.e., data sets that are generated during execution of the program, but do not yet exist) and those defining old (existing) data sets. Old uncataloged data sets can exist only on private volumes.

To define a new data set that is to be written on a public volume, the user may use the DDNAME, DSNAME, SPACE, DSORG, and LABEL operands. Exactly which fields he uses, other than DDNAME and DSNAME (which are required), depends on the character of the particular data set to be defined.

To define a new data set that is to be written on a private volume, the user must give the DDNAME, DSNAME, UNIT and VOLUME operands. If desired, he may also furnish the DSORG, DISP, SPACE and LABEL operands.

The user defines an old, uncataloged data set as it exists on his private volume. He must use the DDNAME, DSNAME, DISP, VOLUME, and UNIT operands; he may also use the DSORG and LABEL operands. The DCB operand is required to specify tape density for any data set on tape, unless the tape density matches that established at system generation time.

Table F-1. Typical Use of DDEF Operands

	DDNAME	DSORG	DSNAME	DISP	DCB	UNIT	SPACE	VOLUME	LABEL	OPTION
Read cataloged data set	X		X	[X]						
Read uncataloged data set	X	[X]	X	X		X		X	[X]	
Write data set on public volume	X	[X]	X	[X]			[X]			

Table F-1. Typical Use of DDEF Operands (cont'd)

	DDNAME	DSORG	DSNAME	DISP	DCB	UNIT	SPACE	VOLUME	LABEL	OPTION
Write data set on private volume	X	[X]	X	X		X	[X]	X	[X]	
Modify data sets on private volumes	X	[X]	X	X		X		X	[X]	
Concatenate cataloged data sets while reading private volumes (for each concatentated data set except the first in concatentation)	X	X	X	X		[X]		[X]	[X]	X

KEY: [ ] indicates that operand may be used, but is not mandatory.

Examples:

1. Read a cataloged data set.

   ddef ddn,dsname=test1,disp=old

2. Read an uncataloged data set.

   ddef ddn1,vi,dsname=test2,disp=new,unit=(da,2311),volume=(,012300)

3. Write a data set on a public volume.

   ddef ddnz,vp,dsname=test3

4. Write a data set on a private volume.

   ddef ddn3,ps,dsname=test4,unit=(ta,9),volume=(private) or volume=(,005431)

5. Modify any data set of a private volume.

   ddef ddn4,ps,dsname=test5,disp=mod,unit=(ta,9),volume=(,012301)

6. Concatentate cataloged data sets while reading private volumes.

   ddef ddn6,ps,dsname=test6,disp=old
   ddef ddn6,ps,dsname=test7,disp=old,option=conc
   ddef ddn6,ps,dsname=test8,disp=old,option=conc

   The DDEF command also has several special uses.

   - To define an existing job library, the following operands are required:

     DDNAME=data definition name,DSORG=VP,DSNAME=data set name,
     DISP=OLD,OPTION=JOBLIB

     No other operands are required.

- To define a new job library, mandatory operands must be given.

  DDNAME=data definition name,DSORG=VP,DSNAME=data set name,
  OPTION=JOBLIB

  No other operands are necessary.

- To define a data set for dumps, mandatory operands must be given.

  DDNAME=PCSOUT,DSORG=VI,DSNAME=data set name

- To complete the data control block of a data set at execution time, the DCB operand is included. Other operands are included as needed for the particular data set.

- To concatenate data sets (i.e., to define them so that several data sets may be read as if they formed a single data set), the OPTION = CONC operand is included. Other operands are provided by the user as needed for a particular data set. The OPTION = CONC operand must be given in the DDEF command for data sets, except the first-defined member of the concatentation. Each of the remaining data sets in the concatenation must have the same DDNAME as the first-defined data set.

Table F-2. Data Set Organization Requirements

Data Set	Data Set Organization (dsorg)				Comments
	PS	VS	VI	VP	
Any data set on public volume		x	x	x	
Any data set on private volume	x	x	x	x	PS applies to tape or direct-access volumes; VS, VI, and VP apply only to volumes on direct-access devices.
Any member of partitioned data set		x	x		Same partitioned data set may include both VS and VI members, (member must be either VS or VI).
SYSIN data set		x	x		
Language Processing					
Source data set for language processing			x		Line data set only; if source data sets are entered from terminal, line data set is automatically built.
Source statements stored as part of SYSIN data set		x	x		Line data set will be built from source statements.
Object module produced by language processor		x			Object module automatically becomes member of most recently defined job library, if any, or of user's library (USERLIB)
Job library				x	
Listing data set produced by language processor			x		
Input/Output					
PCSOUT data set			x		
Input to WT		x	x		
Input to PRINT	x	x	x		
Input to PUNCH		x	x		
Special Command Usage					
Data set for CDD			x		Line data set only.
Data set for LINE?			x		Line or language processor listing data set only.
Data set created by DATA		x	x		User option; if VI, must be line data set.
Data set created by MODIFY			x		User option determines whether VI is line data set.

The status of the current line pointer (CLP), upon completion of the text editing commands is:

1. LOCATE - If the input string is found in a line, the CLP is set to that line's key; if not found, the CLP is set to the last line in the data set plus 100. If no arguments are given, the latter also occurs.

2. CORRECT - The CLP is set to the next line after N2, or to N2 if that is the last line in the data set.

3. NUMBER - The CLP is set to the limiting line key (i.e., the line after N2, or to N2's key plus 100 if that is the last line in the region).

4. STET - Not changed.

5. LIST - The CLP is set to the line number after N2. If N2 is last line in data set, the CLP is set to N2 plus 100 and the last two digits are set to 0.

6. EXCISE - The CLP is set to the first line deleted; if no lines are deleted, to the value specified in N1.

7. REVISE - The CLP is set to the number of the last data line entered plus the current increment (if not input, defaulted to 100).

8. EDIT - The CLP is set to last line number in data set plus 100.

9. UPDATE - Not changed.

10. REGION - The CLP is set to last line in region plus 100. If region name is not in data set, the CLP is set to the new region name with a line number of 100.

11. INSERT - The CLP is set to N1 plus increment (defaulted to 100). If N1 is defaulted, the CLP is updated by the increment.

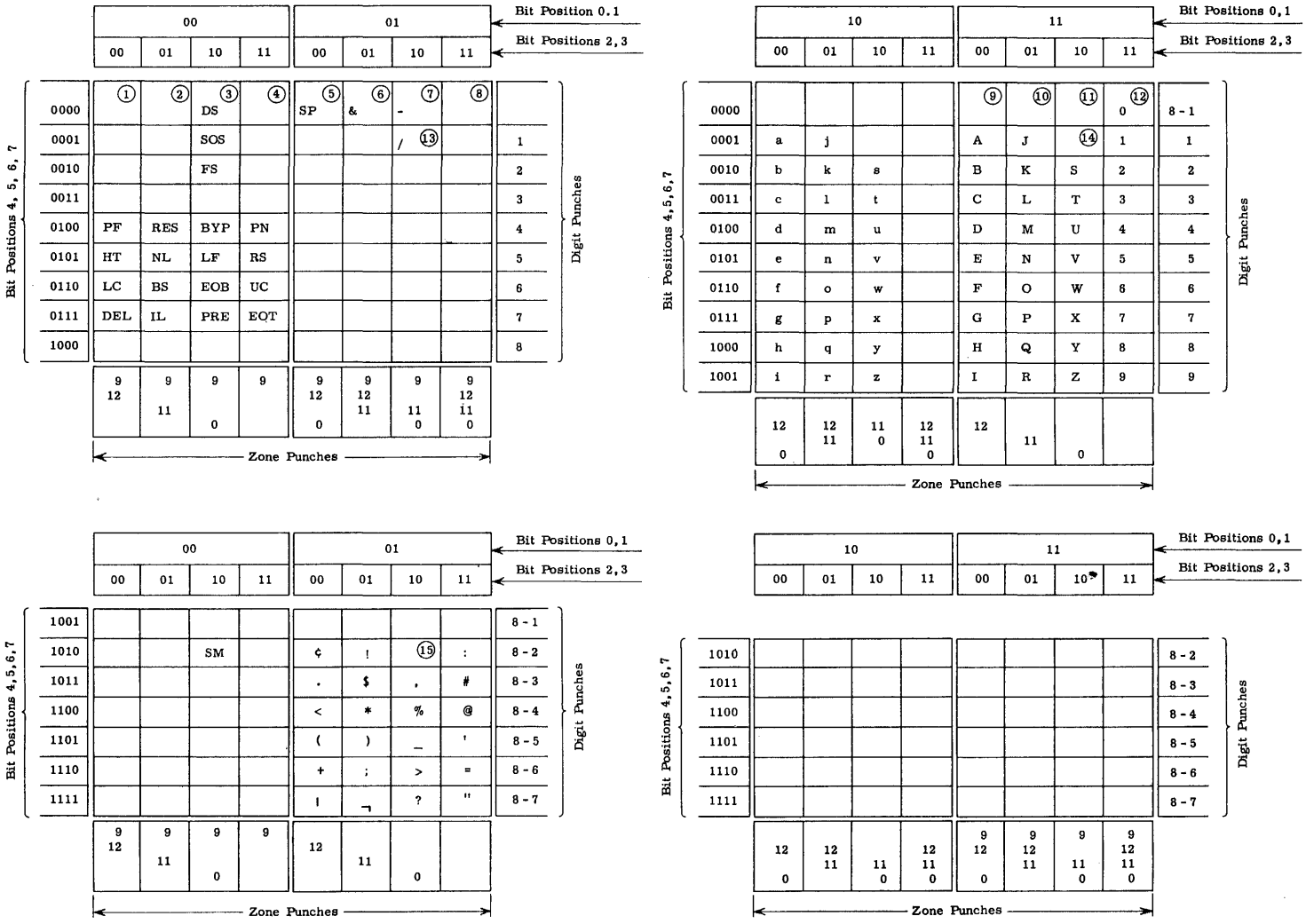
12. CONTEXT - The CLP points to the line following the last line searched (N2).

13. EXCERPT - The CLP points to the next unused line number following the last-included line.

APPENDIX H: EBCDIC CHARTS

The numbering convention for the bit positions within a byte is: 01234567.

The chart below shows bit positions, bit patterns, hole patterns (card), graphic characters, and control characters.



Bit Positions 4,5,6,7	00 00	00 01	00 10	00 11	01 00	01 01	01 10	01 11	Digit Punches
0000	①	②	③ DS	④	⑤ SP	⑥ &	⑦ -	⑧	
0001			SOS				/ ⑬		1
0010			FS						2
0011									3
0100	PF	RES	BYP	PN					4
0101	HT	NL	LF	RS					5
0110	LC	BS	EOB	UC					6
0111	DEL	IL	PRE	EQT					7
1000									8
Zone Punches	9 12	9 11	9 0	9	9 12 0	9 12 11	9 11 0	9 12 11 0	

Bit Positions 4,5,6,7	10 00	10 01	10 10	10 11	11 00	11 01	11 10	11 11	Digit Punches
0000					⑨	⑩	⑪	⑫ 0	8-1
0001	a	j			A	J	⑭	1	1
0010	b	k	s		B	K	S	2	2
0011	c	l	t		C	L	T	3	3
0100	d	m	u		D	M	U	4	4
0101	e	n	v		E	N	V	5	5
0110	f	o	w		F	O	W	6	6
0111	g	p	x		G	P	X	7	7
1000	h	q	y		H	Q	Y	8	8
1001	i	r	z		I	R	Z	9	9
Zone Punches	12 0	12 11	11 0	12 11 0	12	11	0		

Bit Positions 4,5,6,7	00 00	00 01	00 10	00 11	01 00	01 01	01 10	01 11	Digit Punches
1001									8-1
1010			SM		¢	!	⑮	:	8-2
1011					.	$	,	#	8-3
1100					<	*	%	@	8-4
1101					(	)	_	'	8-5
1110					+	;	>	=	8-6
1111					|	¬	?	"	8-7
Zone Punches	9 12	9 11	9 0	9	12	11	0		

Bit Positions 4,5,6,7	10 00	10 01	10 10	10 11	11 00	11 01	11 10	11 11	Digit Punches
1010									8-2
1011									8-3
1100									8-4
1101									8-5
1110									8-6
1111									8-7
Zone Punches	12 0	12 11	11 0	12 11 0	9 12 0	9 12 11	9 11 0	9 12 11 0	

① 12-0-9-8-1
② 12-11-9-8-1
③ 11-0-9-8-1
④ 12-11-0-9-8-1
⑤ No Punches
⑥ 12
⑦ 11
⑧ 12-11-0
⑨ 12-0
⑩ 11-0
⑪ 0-8-2
⑫ 0
⑬ 0-1
⑭ 11-0-9-1
⑮ 12-11

CONTROL CHARACTERS

PF	Punch off	BS	Backspace	PN	Punch on
HT	Horizontal tab	IL	Idle	RS	Reader stop
LC	Lower case	BYP	Bypass	UC	Upper case
DEL	Delete	LF	Line feed	EOT	End of transmission
RES	Restore	EOB	End of block	SM	Set mode
NL	New line	PRE	Prefix	SP	Space
DS	Digit select	SOS	Start of significance	FS	Field separator

SPECIAL GRAPHIC CHARACTERS

¢	Cent sign	*	Asterisk	>	Greater-than sign
.	Period, decimal point	)	Right parenthesis	?	Question mark
<	Less-than sign	;	Semicolon	:	Colon
(	Left parenthesis	¬	Logical NOT	#	Number sign
+	Plus sign	-	Minus sign, hyphen	@	"At" sign
|	Vertical bar, logical OR	/	Slash	'	Prime, apostrophe
&	Ampersand	,	Comma	=	Equal sign
!	Exclamation point	%	Percent sign	"	Quotation Mark
$	Dollar sign	_	Underscore		

Examples	Type	Bit Pattern Bit Positions 01 23 45 67	Hole Pattern Zone Punches	Digit Punches
PF	Control character	00 00 01 00	12-9-4	
%	Special graphic	01 10 11 00	0-8-4	
R	Upper case	11 01 10 01	11-9	
a	Lower case	10 00 00 01	12-0-1	
	Control character, function not yet assigned	00 11 00 00	12-11-0-9-8-1	

Operation	Operands
ABEND	

Operation	Operand
ASM	NAME = module name [,STORED={Y | N [,LINCR=(first line number, increment)]}] [,MACROLIB=({data definition name of symbolic portion, data definition name of index portion}[,...])] [,VERID=version identification] [,ISD={Y|N}] [,SYMLIST={Y|N}] [,ASMLIST={Y|N}] [,CRLIST={Y|N}] [,STEDIT={Y|N}] [,ISDLIST={Y|N}] [,PMDLIST={Y|N}]

Operation	Operands
AT	instruction location [,...]

Operation	Operands
BACK	DSNAME = data set name

Operation	Operands
BRANCH	INSTLOC = instruction location

Operation	Operands
BUILTIN	NAME = command name [,EXTNAME=bpkd macro name]

Operation	Operands
CALL	[NAME = entry point name [,module parameters]]

Operation	Operands
CANCEL	BSN = batch sequence number

Form 1

Operation	Operands
CATALOG	DSNAME = current data set name [,STATE = {N|U}] [,ACC={R|U}] [,NEWNAME = new data set name]

Form 2

Operation	Operands
CATALOG	GDG = generation data group name, GNO = number of generations [,ACTION={A|O}] [,DISP={E|S}]

Operation	Operands
CDD	DSNAME = data set name [, {data definition name / (data definition name,...)}]

Operation	Operands
CDS	DSNAME1 = current data set name, DSNAME2 = new data set name [,DISP={E|S}] [,BASE = first line number, INCR = increment]

Operation	Operands
CONTEXT	[N1 = starting position] [,N2 = ending position] , STRING 1 = search string [,STRING2=replacement string]

Operand	Operands
CORRECT	[N1 = starting line] [,N2 = ending line] [, SCOL = first column] [,*$@% = replacement correction characters]

Operation	Operands
DATA	DSNAME = data set name [,RTYPE = I [,BASE = first line number,INCR = increment]]

DATA Descriptor Card

Col.
3

Operands
DATASET, data set name,[format],[starting number],[ending number], [LINE], [{ACCEPT / SKIP / END}]

Operation	Operands
DDEF	DDNAME = data definition name [,DSORG ={VI | VS | VP}] ,DSNAME = data set name

Operation	Operands
DEFAULT	{operand = [value]} [,...]

Operation	Operands
DELETE	DSNAME = data set name

Operation	Operands
DISABLE	

Operation	Operands
DISPLAY	data field name [,...]

Operation	Operands
DSS?	[NAMES ={data set name / (data set name,...)}]

Operation	Operands
DUMP	data field name [,...]

Operation	Operands
EDIT	[SOURCE = data definition name] [,MNAME = member name]

Operation	Operands
ENABLE	

Operation	Operands
END	

Operation	Operands
ERASE	DSNAME = data set name

Operation	Operands
EXCERPT	DDNAME = data definition name [.member name] [,RNAME = region name] [,N1 = starting line [,N2 = ending line]]

Operation	Operands
EXCISE	[N1 = starting line] [,N2 = ending line]

Operation	Operands
EXECUTE	DSNAME = data set name

Operation	Operands
EXPLAIN	[{word / ORIGIN / TEXT,message identification / RESPONSE}]

Operation	Operands
FTN	NAME = module name [,STORED = {Y / N [,LINCR = (first line number, increment)]}] [,VERID = version identification] [,ISD = {Y | N}] [,SLIST = {Y | N}] [,OBLIST = {Y | N}] [,CRLIST = {Y | N}] [,STEDIT = {Y | N}] [,MMAP = {Y | N}] [,BCD = {Y | N}] [,PUBLIC = {Y | N}]

Operation	Operands
GO	

Operation	Operands
IF	condition

Operation	Operands
INSERT	[N1 = preceding line][,INCR = increment]

Operation	Operands
LINE?	DSNAME=data set name [, {line number / (first line number, last line number)} [,...]]

Operation	Operands
LIST	[N1 = starting position][,N2 = ending position]

Operation	Operands
LNK	NAME = module name [,STORED = {Y | N [,LINCR = (first line number,increment)]}] [,LIB = data definition name of library][,VERID = version identification][,ISD = {Y | N}][,PMDLIST = {Y | N}]

Operation	Operands
LOAD	[NAME = entry point name]

Operation	Operands
LOCATE	[N1 = starting position][,N2 = ending position] [,STRING = character string]

Operation	Operands
LOGOFF	

Operation	Operands
LOGON	user identification,[charge number],[confirmation], [message option],[password]

Operation	Operands
MODIFY	SETNAME = data set name [,CONF = R] [,LRECL = record length,KEYLEN = key length,RKP = key displacement,RECFM = {V | F}]

Operation	Operands
NUMBER	[N1 = starting line][,N2 = ending line] [,BASE = base number][,INCR = increment]

Operation	Operands
PERMIT	DSNAME = {data set name | *ALL} [,STATE = {N | U}] [,ACCESS = access qualification] [,USERID = {(user identification[,...]) | *ALL}]

Operation	Operands
POD?	DSNAME = data set name [,H][,A]

Operation	Operands
PRINT	DSNAME = data set name [,STARTNO = first byte position] [,ENDNO = last byte position] [,PRTSP = {EDIT | {1 | 2 | 3}[,HEADER=H][,LINES=lines per page][PAGE=P]}] [,ERASE = ERASE][,ERROROPT= {ACCEPT | SKIP | END}] [,FORM = paper form]

Operation	Operands
PROCDEF	NAME = procedure name

Operation	Operands
PROFILE	[CSW = {N | Y}]

Operation	Operands
PUNCH	DSNAME = data set name [,CBIN = BINARY] [,STARTNO=first byte position][,ENDNO=last byte position] [,STACK={1|2|3} EDIT][,ERASE=ERASE][,FORM=paper form]

Operation	Operands
QUALIFY	MNAME=[link-edited module name.] object-module name

Operation	Operands
REGION	[RNAME = region name]

Operation	Operands
RELEASE	DDNAME=data definition name[,DSNAME=data set name]

Operation	Operands
REMOVE	statement number [,...]

Operation	Operands
REPEAT	

Operation	Operands
REVISE	[N1 = starting line][,N2 = ending line] [,INCR = increment]

Operation	Operands
RUN	[LOC = entry point name]

Operation	Operands
SECURE	{ (TA = number of devices[,type of device]) / (DA = number of devices[,type of device]) } [,...]

Operation	Operands
SET	{data location = value}[,...]

Operation	Operands
SHARE	DSNAME = data set name, USERID = owner's user identification [,OWNERDS = {owner's data set name | *ALL}]

Operation	Operands
STET	

Operation	Operands
STOP	

Operation	Operands
SYNONYM	{term = [character string]} [,...]

Operation	Operands
TIME	[MINS = minutes]

Operation	Operands
TV	DSNAME1 = tape data set name [,DSNAME2 = vam data set name]

Operation	Operands
UNLOAD	[NAME = entry point name]

Operation	Operands
UPDATE	

Operation	Operands
VT	DSNAME1 = vam data set name [,DSNAME2 = tape data set name]

Operation	Operands
VV	DSNAME1 = current data set name [,DSNAME2 = new data set name]

Operation	Operands
WT	DSNAME = current data set name,DSNAME2 = tape data set name [,VOLUME = tape volume number] [,FACTOR = blocking factor] [,STARTNO = first byte position] [,ENDNO = last byte position] [,PRTSP = {EDIT / {1 / 2 / 3}} [,HEADER=H] [,LINES=lines per page] [,PAGE=P]}] [,ERASE = ERASE]

Operation	Operands
ZLOGON	

READER'S COMMENT FORM

IBM System/360 Time Sharing System
Command System User's Guide

C28-2001-2

- Your comments, accompanied by answers to the following questions, help us produce better publications for your use. If your answer to a question is "No" or requires qualification, please explain in the space provided below. Comments and suggestions become the property of IBM.

	Yes	No
• Does this publication meet your needs?	☐	☐
• Did you find the material:		
Easy to read and understand?	☐	☐
Organized for convenient use?	☐	☐
Complete?	☐	☐
Well illustrated?	☐	☐
Written for your technical level?	☐	☐

- What is your occupation? _____
- How do you use this publication?

As an introduction to the subject?	☐	As an instructor in a class?	☐
For advanced knowledge of the subject?	☐	As a student in a class?	☐
For information about operating procedures?	☐	As a reference manual?	☐

Other _____

- Please give specific page and line references with your comments when appropriate. If you wish a reply, be sure to include your name and address.

COMMENTS:

YOUR COMMENTS PLEASE . . .

This publication is one of a series which serves as reference for systems analysts, programmers and operators of IBM systems. Your answers to the questions on the back of this form, together with your comments, will help us produce better publications for your use. Each reply will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Please note: Requests for copies of publications and for assistance in utilizing your IBM system should be directed to your IBM representative or to the IBM sales office serving your locality.

Fold Fold

FIRST CLASS
PERMIT NO. 34
YORKTOWN HTS., NY

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN U. S. A.

POSTAGE WILL BE PAID BY

IBM Corporation
PO Box 344
2651 Strang Boulevard
Yorktown Heights, N.Y. 10598

ATTN: Time Sharing System/360
Programming Publications Dept. 561

Fold Fold

Printed in U.S.A. C28-2001-2

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]

Printed in U.S.A. C28-2001-2

IBM®

International Business Machines Corporation
Data Processing Division
112 East Post Road, White Plains, N.Y. 10601
[USA Only]

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
[International]