File No. S360-36
GC28-2001-9

IBM Systems Reference Library

IBM Time Sharing System

Command System User's Guide

This is a reference book for users of the commands that are part of the IBM Time Sharing System (TSS). The command system gives the user the ability to (1) construct, execute, and debug programs; (2) create, modify, copy, and share data sets; (3) enter data into and retrieve data from the system; and (4) modify and add to the system-supplied commands.

Commands that are used exclusively by the system operator, system manager, system administrator, and system programmer are not presented in this book.

Three types of information make up the major part of this reference: basic information for the new user, examples, and command descriptions. Part II contains seven sections, and each section describes a different category of commands. Introductory material in Part II is provided to give the new user of the command system general knowledge of the commands. Part III contains format illustrations and descriptions of the commands. Parts II and III also contain examples that show ways of using the commands.

Before reading Command System User's Guide, you should have general knowledge of TSS. For an introduction to TSS see IBM Time Sharing System: Concepts and Facilities, GC28-2003. If you enter commands through a terminal, you should be familiar with the terminal. See IBM Time Sharing System: Terminal User's Guide, GC28-2017 for instructions on operating the IBM 2741 Communications Terminal and the IBM 1052 Printer-Keyboard. A list of publications related to TSS appears in the IBM Time Sharing System: Addendum, GC28-2043 .

Tenth Edition (August 1976)

This is a major revision of, and makes obsolete, GC28-2001-8. Extensive editorial and technical changes have been made in this edition. Among the modifications to the system that are reflected in this publication are the following:

- Addition of the following commands:

BLIP	EJECT	FTNH	CDC	PLIOPT
BLIP?	FILEDEF	HASM	OSDD?	SPACE
COBOL	FILEREL	LL	OSRUN	TRANSLAT

- Changes to the SCRATCH and HOLD options of the RELEASE command

- Clarified unit type parameters for several commands

This edition is current with Release 2.0 of the IBM Time Sharing System/370 (TSS/370), and remains in effect for all subsequent versions or modifications of TSS unless otherwise noted. Significant changes or additions to this publication will be provided in new editions of Technical Newsletters.

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, Time Sharing System, Dept. 80M, 1133 Westchester Avenue, White Plains, New York 10604.

PREFACE

This is a general-purpose reference manual for the IBM Time Sharing System (TSS) command system. Since users of this book have varying degrees of knowledge about the command system, there are several levels of information in it. The book is organized as follows:

- Part I explains the basics of the command system (for example, what a command statement is) and the method of describing commands.
- Part II is divided into six sections, each describing one group of commands. This contains general information about the commands.
- Part III contains format illustrations and descriptions of the commands. The commands appear in alphabetical order. Examples of their use are provided.
- The appendixes provide supplementary reference material, for example, bulk I/O procedures, system-supplied tables and default values, special codes for printer and punch control, and detailed information about some commands.

If you are a new user of the command system, you should read Part II to get background knowledge about the commands. If, in addition, you are a new user of TSS, read IBM Time Sharing System: Concepts and Facilities, GC28-2003 an introduction to the system. When you enter commands into the system, enter them as shown in the format illustrations in Part III. If you are entering commands at a terminal and are not familiar with the terminal, read IBM Time Sharing System: Terminal User's Guide, GC28-2017. You may have to review the command descriptions and examples that are in Part III several times while you are learning the system.

As you become familiar with the command system, you may use only the format illustrations in Part II. If you need a quick reference, you may go immediately to Appendix G.

Commands that are used exclusively by a system operator, administrator, manager, or programmer are not described in this book. For information about these commands see IBM Time Sharing System: Operator's Guide, GC28-2033, IBM Time Sharing System: Manager's and Administrator's Guide, GC28-2024, and IBM Time Sharing System: System Programmer's Guide, GC28-2008.

A publication list appears in IBM Time Sharing System: Addendum, GC28-2043. Some other publications in the IBM Time Sharing System library that you may need to use, in addition to those that are listed above, are:

System Messages, GC28-2037

Quick Guide for Users, GX28-6400

Assembler Language, GC27-2000

Assembler Programmer's Guide, GC28-2032

Assembler User Macro Instructions, GC28-2004

Introducing TSS: A Primer for FORTRAN Users, GC20-2048

FORTRAN Programmer's Guide, GC28-2025

Linkage Editor, GC28-2005

PL/I Programmer's Guide, GC28-2049

CONTENTS

TABLES

FIGURES

PART I: THE COMMAND SYSTEM

The command system is the principal means with which you communicate with the IBM Time Sharing System (TSS). With the command system, you can create, execute, and debug your programs; you can create, alter, and destroy collections of data (known as data sets); and you can modify system commands or write your own commands. Each command acts as an instruction to the system; that is, it tells the system what operation you want performed and on which data you want to operate.

The command system can be used in two modes: conversational and nonconversational. In conversational mode you interact with the system at your terminal. As you enter each command, it is executed by the system. In nonconversational mode, the system executes one or more commands that exist in a prestored data set. Each command is executed as it is entered into the system from this data set, but there is no interaction between you and the system through the terminal. In either mode, if you have not designated that you are finished using the system, you are prompted for another command.

The system commands have been grouped into seven categories. These categories and the sections in which they are described in Part II are listed below:

- Task management (see Section 1)
- Data management (see Section 2)

  Bulk output (see Section 2)
- Program management (see Section 3)
- Command creation (see Section 4)
- Message handling (see Section 5)
- User profile (see Section 6)
- Program Product Language Interface (see Section 7)

Task management commands allow you to initiate, control, and terminate the processing of your task. These commands do not manipulate data, nor do they create programs; they are used to start and to stop execution of your task.

Data management commands allow you to create and manipulate data sets. These data sets can contain programs to be compiled, data to be maintained, or commands to be executed.

Bulk output commands allow you to initiate output operations on system printers or punches.

Program management commands allow you to initiate and control the execution of your programs, to initiate compilation or assembly of data sets created with data management commands, and to debug your programs after they have been compiled or assembled.

Command creation commands allow you to create your own commands to supplement or replace system commands.

Message handling commands allow you to create your own message file from which messages are issued to your task. You can supplement the system's message file, create your own new messages, and display messages for review or clarification.

User profile management commands allow you to control the environment in which your task executes. You can alter values for operands; you can change the translation tables; and you can choose whether to make your changes permanent or temporary.

The Program Product Language Interface (PPLI) allows you to execute the OS/VS program product compilers via interface modules.

Note: Unless otherwise noted, the commands described in this book may be used in either conversational or nonconversational operations.

This publication describes the command system for use in doing productive work. The system maintainence commands are described in System Programmer's Guide, Operator's Guide, and Manager's and Administrator's Guide.

COMMAND FORMAT AND NOTATION

The basic format of a command is:

Operation	Operand
command name	one operand; several operands delimited by commas or tabs; 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 or tabs. Blanks may be used between operands, in addition to the delimiter, but they are ignored by the system. For example:

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

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

Note: In the examples throughout this publication, commas are used as operand separators.

COMMAND STATEMENT

A command statement is one command or a series of commands that are separated by semicolons. The system recognizes a command statement as one SYSIN record. Normally, one command statement is written on one line; however, when one command statement is written on more than one

line, you must end each line that is being continued with a hyphen. You may comment a command, but the comment must be separated from the command by a semicolon and the comment must be written between single quotation marks. (If several commands are used in the command statement, there must be a semicolon before the comment and another semicolon afterthe comment.) Comments do not affect execution. Comments can be used in places other than on a command statement. You may begin a comment after a system underscore, as follows:

```
_'this is a comment'
```

There are three types of command statements: dynamic, immediate, and conditional. A dynamic command statement contains an AT command, which specifies the location where the commands in the statement are executed. An immediate command statement does not contain an AT command and is executed when it is entered into the system. A conditional command statement (dynamic or immediate) contains an IF command, and the part of the statement following IF is executed only when the condition stipulated by IF is true. The following are examples of immediate (1, 2, and 3), dynamic (4), and conditional (5) command statements:

```
1.  delete myds; 'erase the catalog entry for myds'; cancel 3219;-
    'eliminate print task'

2.  execute data1; catalog data2,u,u;logoff

3.  wt dsname=abc,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; 'test variable x'
```

The commands are described and some examples of the use of commands are given in Part III. See also "Use of Command Statements" in Section 3 of Part II.

OPERAND REPRESENTATION

The system can determine the value of a specific operand in one of two ways: (1) from the position of that operand within a series of operands or (2) from a descriptive keyword preceding the operand value. When positional operands are used, they must appear in the order that is shown in the command format illustration. If a positional operand is omitted and another positional operand is written following the omitted operand, the delimiter (that is, comma or tab) that would have followed the omitted operand must be used 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 is shown in the illustration in all-capital letters, and value is the actual value of the operand. Delimiters are not required to indicate omitted keyword operands. When you enter keyword operands in positional notation, you

can omit the keyword. You then follow the rules for entering operands in positional notation.

Keyword and positional representation of operands may be used simultaneously in the same operand field. 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,B=y,C=z    A=x,y,z    A=x,B=y,z    x,C=z,B=y    x,C=z    B=y
```

When the keyword form is used, the same keyword operand can be repeated several times in the operand string; however, the system uses the last value specified in the operand string for a given operand. For example:

```
command a=x,b=y,a=u,c=z,a=p
```

In this example, the keyword operand A takes a final value, A=P. If a value is given to the same operand in both keyword and positional representation, the last value encountered in the command statement is used when the command is executed. For example:

```
x,c=1,y,b=u
```

This command executes as if the values were:

```
x,u,y
```

Operands are resolved from left to right; that is, the last value encountered for a given operand is used when the command is executed. Again, if a command format shows three keyword operands:

Operation	Operand
COMMAND	A=term,B=value,C=name

the command can be entered as:

```
command x,y,z,b=p
```

The operand specified by B is resolved as B=p. The command executes as:

```
command x,p,z
```

A self-defining keyword has all the properties of a normal keyword. In addition, the keyword may appear in a command by itself, without an equal sign and value; in this case the user will be passed the single character 'Y'. If the keyword is entered by itself but prefixed with 'NO', the user will be passed the single character 'N'. For example, the following two commands pass the same parameters to the user:

```
PROFILE  TASK=N
PROFILE  NOTASK
```

The examples are a guide; they do not contain all possibilities for these operands. Operand resolution is described more fully in Section 4 of Part II.

COMMAND FORMAT ILLUSTRATIONS

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

USE OF METASYMBOLS

To make the operands in the format illustrations clear, four metasymbols are used:

Name	Symbol	Use
braces	{ }	delimit syntactical units (one or more operands) that may be repeated; delimit alternatives.
brackets	[ ]	delimit optional names and operands, or both, in the appropriate field.
vertical stroke	|	separates choices for the operand; for example, {A|B} denotes that, for the syntactical unit enclosed by the braces, either A or B may be chosen, but not both. {A|B|C} denotes that a choice must be made between A, B, and C. Alternatives may also be indicated by aligning the choices vertically within the braces: $\left\{\begin{matrix}A\\B\end{matrix}\right\}$
ellipsis	...	indicate that the preceding syntactical unit may be repeated one or more times. If there is a system limit to the number of repetitions permitted, this is given in the operand list that follows the format illustration.

OPERATION FORMAT

To distinguish command names in the format illustrations, uppercase letters are used. The user may enter command names in either uppercase or lowercase letters, depending on his mode of input (see "SYSIN Device and Character Control," in Section 1 of Part II). In folded mode (that is, uppercase letters and lowercase letters are equivalent), he may use both. Unless specified by the user, this is the normal mode of keybcard input. In full EBCDIC mode (that is, uppercase and lowercase letters are differentiated by the system), he must use uppercase letters. The operation must be separated from the operand by one or more spaces or by a tab. A detailed description of how to enter commands is given in Section 1 of Part II, under "Communicating with the System."

OPERAND FORMAT

Within the operand field of the format illustration, the word or phrase that is used to identify each operand is written entirely in lowercase letters. For positional operands, only the lowercase word or phrase appears; for keyword operands, the keyword (to the left of the equal sign) is in uppercase letters, and the keyword descriptor (to the right of the equal sign) is in lowercase 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 must be written exactly as shown in the format illustration. Coded values always appear in format illustrations as numbers or uppercase 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 (or tabs) must always be used 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 format illustrations.

OPERAND DESCRIPTIONS

Detailed information about writing each operand is given in a list following every format illustration. At the end of the operand description information that appears under the heading "Specified as" describes the valid specifications for the operand, and information under the heading "System default," describes the system's action if the operand is omitted. System default is not shown if the system's default value is null.

COMMAND 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 are omitted. "Cautions" are statements of warning to the user about difficulties he may have in using the command. "Cautions" appear only where applicable.

In the examples of command usage that follow the command description you are given a brief description of what is being accomplished. The example shows the user's input and the system's response.

GENERAL TERMS

These general terms are used in many of the command descriptions in Part III. (More terms, referring to a specific functional group of commands, are defined under "General Terms" in Section 2 of Part II.)

cataloged
: a data set is cataloged when its name and other pertinent information is entered in the user's catalog. All VSAM, VISAM, and VPAM data sets are automatically cataloged when they are created. All others are cataloged via the CATALOG or EVV command.

data definition name (DDNAME)
: 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 (DSNAME)
: the name used to identify a data set. A data set name consists of one or more simple names. Each simple name has from 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. Therefore, the maximum number of simple names is 18.

Fully qualified data set name: identifies one specific data set; it includes all simple names (that is, 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 of a command or to an implied operand. This value is used when the operand is 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, a DDEF macro instruction, or a CDD command that results in execution of a prestored DDEF command.

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 that is 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 data set or region data set. Also, line may refer to a unit of information entered from a terminal. In this context, a line consists 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 from 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.

source list
a string of commands or program calls used as input to the system.

volume identification
the identification assigned to a specific volume. The volume identification consits of from one to six alphameric characters.

PART II: USE OF COMMANDS

The system-supplied commands are grouped into seven categories. Each group has a different function. The groups are:

- Task management
- Data management
- Program management
- Command creation
- Message handling
- User profile
- Program Product Language Interface (PPLI)

The commands in each group are discussed in the sections that follow. Detailed descriptions of the commands are presented in Part III.

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

The user's task comprises all the work done by and for the user. It is initiated by a LOGON command and is terminated by a LOGOFF command. However, certain conditions, such as an abnormal task termination (ABEND), cause the system to issue a LOGOFF command to terminate the current task and a LOGON command to initiate a new task for the user.

The task management commands and their system functions are shown in Table 1.

Table 1. Task management commands and their functions

Command	Function
ABEND	Eliminate the current task; start a new task.
ABENDREG	Display register contents at the time of the most recent ABEND.
BACK	Shift the user's conversational task to nonconversational mode.
BEGIN	Connect the user to an MTT application program.
CANCEL	Terminate execution of a nonconversational task prior to its normal end.
CHGPASS	Alter the user's password.
EXECUTE	Initiate a previously defined nonconversational task.
EXHIBIT	Display the activity of the batch work queue or of a user task.
LOGOFF	Notify the system that the user wants to terminate his task.
LOGON	Identify the user to the system for initiation of his task.
SECURE	Identify the types of I/O devices that are needed for private data sets in a nonconversational task.
TIME	Establish a time limit for execution of the task.
USAGE	Present statistics relating to the user's utilization of system resources.
ZLOGON	Perform, at LOGON, a user's previously defined procedure.

Communicating with the System

The user is known to the system by his user identification, which is assigned to him by an installation administrator using the JOIN command. All the user's data is stored in the system under his identification.

Thus, when the user is connected to the system, the minimum data required to initiate communication is his user identification.

Resource Control

When the user is joined to TSS, he is assigned a user limits table, which controls the allocation of system resources for his use. The user is limited to the amount of CPU time, connect time, pages of permanent and temporary storage, unit-record devices, direct access devices, magnetic tape devices, bulk I/O, and tasks. These rations are imposed either for a specific time period or for the cumulative time that the user is joined to the system. The limits established for a user are determined by his installation. The user can examine his usage of resources with the USAGE command (see Part III).

CONVERSATIONAL MODE

CONVERSATIONAL TASK INITIATION

After turning on his terminal and dialing the system, the user initiates his task by entering a correct LOGON command. He also has the option of connecting to an already running task with the BEGIN command. He may enter his commands through his terminal keyboard, or card reader, to direct execution of his task.

SYSIN: This name designates the source of 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 dialog 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 dialog consists of messages to the user, responses to his command statements, and requests for command statements. The user has control over the length and type of messages he receives. (Details are presented later in Section 5. Message texts appear in this manual as part of the examples. Messages are published in System Messages, GC28-2037. The system issues general information messages and diagnostic messages that inform 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.

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. Uppercase and lowercase notations in this publication are illustrative; command statements may be entered in either form.

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 is continued on the next line.

Command statements that are entered through the terminal card reader can utilize free-form format (that is, 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 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 may not be printed correctly at the terminal printer, even though they are correctly transmitted to the system.

REQUEST FOR NEXT COMMAND STATEMENT: The system informs the terminal user that it is ready to accept his next command statement by printing a prompt character that, initially, is an underscore character (_) in the first character position of a new line. (The system backspaces one space so that the first character you enter is above the underscore.) When the terminal card reader is being used to enter input, the system signals that it is ready for the next card, or the next command on a card, by printing the underscore.

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

When the user initiates a conversational task, the system checks the value of the ALPHABET operand in the user's profile (see Section 6). Initially, its value is 1, which indicates folded mode. The user can type any lowercase letter, and the system converts it to uppercase. The special characters, , ", !, @, #, and $, are valid alphabetic characters in either mode. The system accepts 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 reads all the cards or reads cards until the user presses the ATTENTION key or until a K, KA, or KP 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.

Table 2. Commands for SYSIN device and character set selection

Command	Function
C	Transfer control to the card reader; if the keyboard mode was KA, CA is the new mode; if KB was the keyboard mode, CB is the new mode.
CA	Transfer control to the card reader and convert card input from 1057 Card-Punch code to EBCDIC.
CB	Transfer control to the card reader and convert card input from 029 punch code to EBCDIC.
K	Transfer control to the keyboard; if the character set used during card reader input was CA, KA is the new mode; if CB was the card reader mode, KB is the new mode.
KA	Transfer control to the keyboard and use the full EBCDIC character set. Can be used to change the ALPHABET operand without transferring control.
KB	Transfer control to the keyboard and use the folded character set. Can be used to change the ALPHABET operand without transferring control.

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. Lastly, 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 the 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 have the same effect whether they are entered in one statement or in individual statements. For example:

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

produces 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 the CALL command had been entered incorrectly and was canceled by the system, the user would not be able to correct the command until the other two commands were executed. In the second example, he would be able to correct his error before PRINT and DELETE were executed.

COMMAND STATEMENT RESOLUTION: When a command statement is entered, the system goes through this sequence of events to resolve the contents of the statement.

1. The system searches a list of synonyms to see if one exists for the specified command name. If one does, the system replaces the specified name. This step is repeated until no further synonym is found.

2. The system searches the user's procedure library, which contains commands he has written, to see if the command name exists there. If it does, the system performs the action described in Item 5. If it does not, the system performs the action described in Item 3.

3. The system searches the system's procedure library, which contains system-supplied commands. If the command name is there, the system performs the action described in Item 5. If not, the system does Item 4.

4. The system assumes that what was entered was a module name, not a command name. A direct program call is executed.

5. The system resolves the operands.

   a. When keywords are specified, the list of synonyms is searched. If a synonym is found, it is inserted in place of the specified keyword.

   b. If explicit operand values are specified, they are used; if not, the system does Item c.

   c. A list of user default values is searched; if values exist, they are inserted; if not, the system does Item d.

   d. A list of system default values is searched; if values exist, they are inserted; if not, the system does Item e.

   e. The operand is given a null value.

6. This command is invoked.

Note: The default values initially assumed by the system are described in Appendix C; how to specify defaults and synonyms is discussed in Section 6.

CONVERSATIONAL TASK INTERRUPTION

The user can interrupt execution of his conversational task by pressing the ATTENTION key at his terminal. The system's response depends upon when the interruption occurs (see Table 3).

Table 3. System responses to attention interruptions

System Response to ATTENTION	Explanation of Response
_ (underscore)	The last command in the source list was being executed, but the system was interrupted. (The system is executing privileged code.) The system cleans up before it prints the underscore.
* (asterisk)	Privileged code was interrupted before execution of last command in the source list was started.
! (exclamation)	Nonprivileged code was interrupted; the processing can be resumed if the interrupted source list has not completed execution.

When the user presses the ATTENTION key while privileged code is executing, the system responds with either an underscore or an asterisk. In either case, the user cannot resume execution of the interrupted command. He gets control in the command mode. If an * is received, the user can resume execution of the source list (by pressing the carriage return) at the command following the interrupted command.

When the user interrupts execution of nonprivileged code (the system responds with an exclamation), any remaining commands in the source list and the status of the interrupted program are saved. After receiving the !, the user has several options:

- He can issue the ABEND command to cancel his task;
- He can issue the GO command (or press the carriage return) to resume processing at the point of interruption;
- He can issue some valid command (either a system-supplied command or one of his own making), including direct calls to nonprivileged modules;
- He can use one of the TSS attention handling commands, described below and in Part III of this book.

The ABEND command cancels the task and initiates a new task. All programs are unloaded; all DDNAMEs are released. The user is now in command mode (see ABEND command in Part III).

The GO command gives control to the most recently interrupted program at the point of interruption (see GO command in Part III).

When the user follows an attention interruption with some command other than ABEND or GO, the previously interrupted program is saved for later execution. New command statements are honored. These new commands can be interrupted; their status is saved, too. The user can interrupt, and save, as many as 10 source lists (commands or programs). If he exceeds this number, he receives a diagnostic message.

There are attention handling commands that allow the user to control the processing of interrupted source strings. These commands, described in Part III, are listed below:

- EXIT -- bypass current module or program; process next command in the source list
- PUSH -- save status of an active program
- RTRN -- clean up all current source lists
- STACK -- display names of active programs
- STRING -- display modules not yet processed in the current source list

If the user wants to run sections of his code that will not be affected by an attention interruption, he can set the attention intervention prevention switch (AIPS), as described in Assembler User Macro Instruction. When this switch is set, the system does not interrupt the user's program when one attention interruption is received. However, a simulated attention interruption is made so that the user can later test the AIPS to see where the attention occurred. Setting the AIPS only inhibits a single attention interruption. If a subsequent attention interruption occurs, the program stops executing and the user's keyboard is unlocked.

Macro instructions in the assembler language allow the user to supply his own attention-interruption-handling routines. See also Assembler User Macro Instructions.

CONVERSATIONAL TASK TERMINATION

The user ends his conversational task by entering a LOGOFF command at his terminal, or he may switch his conversational task to nonconversational mode (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. The user can examine the results of processing. He can issue the LINE? command (if the data set containing the results is a line data set) to obtain a listing at his terminal, or he can issue one of the bulk output commands (see "Bulk Output Commands" in Section 2) 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. (See Assembler Programmer's Guide and FORTRAN programmer's Guide.)

SYSOUT: This name designates the output that is produced during execution of the user's task. This output 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 is delivered at the user's terminal. The SYSOUT for a conversational task is not normally recorded by the system in any form. Since SYSIN (see the defintion of this term under "Conversational Task Initiation" earlier in this section) is entered 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 (see below). Any system messages resulting from the execution of the task are received by the user as a 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. The command statements 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 (variable-format or fixed-format records) 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 I/O devices are to be used by the task, the SECURE command must immediately follow the LOGON command.

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 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. After the nonconversational SYSIN data set has been 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 is cataloged, and execution is 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 (see Appendix B).

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

The BSN 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 interaction 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 ignores it and continues 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. Resolution of command statement elements is identical to that described earlier in this section for conversational execution.

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 canceled.

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 is printed as part of SYSOUT for the task.

   The user can give a response to a system prompt even when his task is running nonconversationally. He must default RSVP=Y and he must provide the response in his SYSIN data set. When the prompt is issued, the response is used and the task is not terminated abnormally. There must be a response for each prompt or the task is terminated.

4. A system shutdown terminates all nonconversational tasks. Those initiated by PUNCH, PRINT or WT are restarted when the system resumes operation. No restart is attempted for other nonconversational tasks.

NONCONVERSATIONAL ABEND CONTROL

The system may terminate a nonconversational task if there is some problem that needs to be resolved by a user. Such a case occurs when the nonconversational task executes a command that issues a prompt that requires a response. When this happens, the system terminates the task, and diagnostic messages are printed as part of the SYSOUT data set.

There is a way to control the system's action when this condition arises. The user can provide a special data set that receives control when his nonconversational task would otherwise be terminated. He can, within his task, define a data set (via DDEF) that is cataloged and that contains a series of commands:

```
ddef tskabend,vi,dsname1
```

Notice that the DDNAME of the data set must be TSKABEND. The data set must be cataloged. Rather than terminate a nonconversational task, the system finds the data set defined with the TSKABEND DDNAME; then, the system executes the commands from that data set (the data set should end with a LOGOFF command).

Example: The user has a cataloged data set, named DSNAME1, that contains these commands: PC?, USAGE, and LOGOFF. In his nonconversational task, he defined this data set with the TSKABEND DDNAME. Then, when the nonconversational task is executing, the system finds that the operand of a LINE? command is a nonexistent data set name, NCDS. The system stops executing the SYSIN data set (the nonconversational source list) and begins to execute the commands in the DSNAME1 data set. A diagram that represents the flow of control is shown in Figure 1.

```
NONCONVERSATIONAL TASK          DSNAME1
LOGON NICK

DDEF TSKABEND,VI,DSNAME1        PC?
   .                            USAGE
   .                            LOGOFF
   .
LINE? NODS
   .
   .
   .
LOGOFF
```

Figure 1. Flow of control from a nonconversational task to a data set that continues the task to normal termination

NONCONVERSATIONAL TASK OUTPUT

The user specifies, by commands in his nonconversational SYSIN data sets, 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 a 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 a nonconversational task. There is no way for him to switch from 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 must not begin with a LOGON command (however, it must end with a ICGOFF command).

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

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

If the system accepts the user's request, it establishes the nonconversational task, assigns it a batch sequence number, and eliminates the conversational task from the system. The user's terminal is then available to him, and he can enter a new conversational task with the LOGON procedure.

There are commands that allow the user to manage his data sets. These commands are divided into four groups, as follows:

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

DATA SET MANAGEMENT

The data set management commands are used to identify data sets; to store them in the system and to retrieve them from the system; to share them with other users; to copy and erase them; and to define them for use in the system. The data set management commands and their system functions are shown in Table 4.

TEXT EDITING

Text is edited by using the text-editing commands. These commands manipulate lines of information that are within an existing region data set or line data set, or the commands manipulate lines as they are being entered into a region or line data set. With these commands, the user can simultaneously create and edit data sets; he can correct, insert, and delete lines; he can segment a data set into regions; 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 can nullify previous changes that were made by the commands. The text-editing commands and their system functions are shown in Table 5.

GENERAL TERMS

The following terms are used throughout the discussions on the text editor.

break character
: when the user enters a break character (which can be the system-supplied underscore or some user-supplied substitute character), the system interprets the statement that follows as a command. The break character allows the user to enter commands when the system expects data. However, when the first and second characters of the line are break characters, the usual break-character action does 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 multiple break characters can be put into procedures or data sets.

current line pointer
: is an indicator that is maintained by the text editor. The current line pointer (CLP) is set initially to the value of BASE (which defaults to 100) for empty regions or empty line data sets, and to the first available line in an existing region or line data set. The CLP is advanced through the region or line data set as text-editing commands are executed, always pointing to the next line to be processed. For rules that govern the positioning of CLP, refer to Appendix F.

Table 4. Data set management commands and their functions

Command	Function
CATALOG	Create or alter a catalog entry for a physical sequential data set; alter a VAM catalog entry; create a catalog index for a generation data group; or catalog a data set as a new generation of an existing generation data group.
CDD	Retrieve DDEF commands, which have been prestored in a cataloged or a defined line data set, and process them.
CDS	Duplicate a data set or a member of a VPAM data set.
CLOSE	Closes a user's data sets.
DDEF	Define a data set and describe its characteristics to the system.
DDNAME?	List DDNAMES.
DELETE	Delete one private data set entry from the user's catalog.
DSS?	Present the status of cataloged data sets.
ERASE	Free direct access storage assigned to a private or a public data set and remove its catalog entry from the user's catalog.
EVV	Catalog private VAM data sets.
FILEDEF	Define a dataset, describe its characteristics, and provide the link between TSS and OS/VS datanames.
FILEREL	Delete a data definition established by a previous FILEDEF and disconnect the OS/TSS link.
JOBLIBS	Manipulates DDNAMES.
LTDS	List data set names from a VAM tape.
PC?	Obtain the name, access qualification, and owner's user identification of cataloged data sets.
PERMIT	Authorize or withdraw authorization of other users to access a user's specified data set.
POD?	Display information about members of a VPAM data set.
RELEASE	Delete a data definition established by a previous DDEF command.
RET	Change the catalog attributes of a VAM data set.
SHARE	Allow a user to share data sets belonging to another user who has granted authorization with the PERMIT command.
TV	Retrieve a data set that was written onto tape via the VT command and write the data set into a VAM volume.
VT	Copy a VAM data set onto tape as a physical sequential data set.
VV	Copy a VAM data set on direct access storage.

Table 5. Text-editing commands and their functions

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

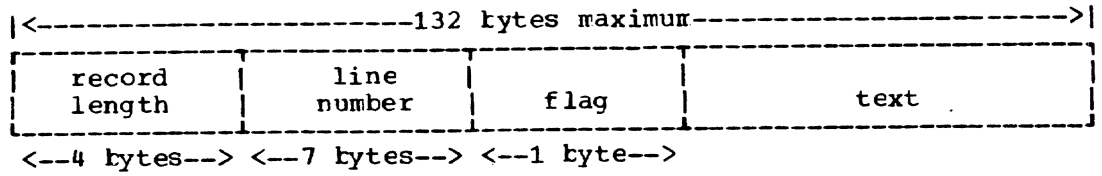
In all text-editing commands, where N1 is an operand, except as indicated in the operand descriptions, the current value of CLP may be specified by defaulting N1. The value of CIP may be displayed by issuing:

```
list clp
```

hexadecimal constant
has the form of an X followed by a string that is enclosed within apostrophes; the characters in the qucted string must be a digit (0 through 9) or A, B, C, D, E, or F. There can be no other characters within the apcstrophes. Some examples are:

```
X'01'
X'ABC02'
X''
```

line data set
is a VISAM data set that has this reccrd format:



Key

The record length field, which appears only in variable-length records, and the flag field are used only by the system. Total record length, defined in the DCB subcperand of the DDEF command, cannct exceed 132 bytes. The beginning of a line data set record is the first digit of the line number. The relative key position is four.

line number specificatcn
is a number that represents a line. This number can be expressed either as an absolute (for example, 200 specifies line 0000200) or as a relative value (positive or negative). A relative value indicates a number of lines distant from CLP (for example, +2 identifies the second line after CLP, and -1 designates the line immediately preceding CLP).

N1 and N2 are the keywords for the beginning line number and the ending line number operands that are used with certain commands (for example, CORRECT, INSERT, LIST, REVISE). The values specified for N1 and N2 are resolved by the system according to the following rules:

1. When the operands are in the range of line numbers that exist in the current region or in the data set being processed, the system uses the value that was specified.

2. The following items discuss the commands except INSERT and REVISE.

   a. If N1 is a number greater than the last line of the current region or of the data set, or if N2 is a number less than the first line of the current region or of the data set, the command is canceled and a diagnostic message is issued.

(Note: The diagnostic messages you get depend on the values of BREVITY and LIMEN. See "Implicit Cperands" in Section 6 and Appendix C.)

b. If N1 and N2 are two different numbers and are within the limits of the current region or of the data set, but the line does not exist, the system uses the next-higher-numbered line for N1 and executes the command and uses the next-lower-numbered line for N2 and executes the command.

c. If N1 and N2 specify the same line and the line does not exist, the command is canceled and a diagnostic message is issued.

3. The following items discuss the INSERT and REVISE commands.

a. An INSERT command is executed when N1 is outside the range of the data set or current region; N1 is within the range, but does not exist; or N1 is within the range but, with the increment (assumed or specified), points to a line that does not exist.

b. A REVISE command is executed if the line that is specified does or does not exist within the data set or even if the line is outside the limits of the data set.

Example: The following data set exists. The accompanying commands show what the system prints out after it resolves the values you specify for N1 and N2. (This user has set LIMEN=I.)

```
0000100 N1 AND N2 ARE WITHIN THE RANGE
0000200 N1 IS GREATER THAN THE LAST LINE
0000300 N2 IS LESS THAN THE LAST LINE
0000400 N1=N2, BUT THE LINE DOES NOT EXIST
0000500 N1 NOT= N2 AND THE LINE DOES NOT EXIST
0000600 INSERT COMMAND TESTS
0000700 REVISE COMMAND TESTS
0000900 LINE 8 DOES NOT EXIST
0001000 LAST LINE OF THE DATA SET
```

```
User:  edit linetest
Sys,User:  list 100,200
           0000100 N1 AND N2 ARE WITHIN THE RANGE
           0000200 N1 IS GREATER THAN THE LAST LINE
           CZASP100      CLP SET TO 0000300
           excise 1500
           CZASL500      N1, 1500, BEYOND END OF DATA SET OR REGION
           excise 100,50
           CZASL600      N2, 50, LESS THAN START OF DATA SET OR
                         REGION
           excise 50,200
           list 50,200
           CZASL600      N2, 200, LESS THAN START OF DATA SET OR
                         REGION
           excise 900,1500
           list 800,1500
           CZASL500      N1, 800, BEYOND END OF DATA SET OR REGION
```

Note: This user reinserted the deleted lines (100, 200, 900, 1000).

```
           list 800,last
           0000900 PREVICUS LINE 8 DOES NOT EXIST
           0001000 LAST LINE OF THE DATA SET
           CZASP100      CLP SET TO 0001100
           list 800,800
           CZASP200      CANCELED:  RANGE INVALID
           list 800
```

```
CZASP300      CANCELED:  LINE 800 DOES NCT EXIST
insert 100
CZASG100      CANCELED:  LINE 0000200 ALREADY EXISTS
insert 100,50
0000150 a new line
CZASG050      COMPLETED:  LINE 0000200 ALREADY EXISTS
insert 700
0000800 _insert 1500
0001500 _revise 50
```

(Note: The user issued the following commands -- REVISE 50,50; REVISE 200; REVISE 800; and REVISE 1500. In each case the system responded by prompting with the specified line number.)

```
1500_logoff
```

offset
in three commands -- CONTEXT, LIST, and LOCATE -- the user can specify starting and ending character positions in the N1 and N2 operands. He does this with an one- to four-digit absolute decimal number, enclosed in parentheses, and immediately following the line number. The first character of data is at position 1. For example: N1=700(4) specifies the fourth character of line 700; N2=900(14) specifies the fourteenth character of line 900.

prompting
is done by the text editor following entry of a command that expects data (for example, EDIT, REGION, REVISE, INSERT). When the LINENO operand in the user profile has a value of Y, the text editor issues a line number when it expects data; if LINENO=N, the keyboard is unlocked, but line number prompting does not occur. Following execution of a command that does not expect data (for example, STET, NUMBER, EXCISE) or that completes its execution, the text editor prompts the user for a command statement by issuing an underscore.

The user can inhibit line number prompting by issuing

```
default lineno=n
```

before or during text editing. Initially, LINENO=Y.

Even when line number prompting is inhibited, the text editor inserts the line number in the key of each data set line.

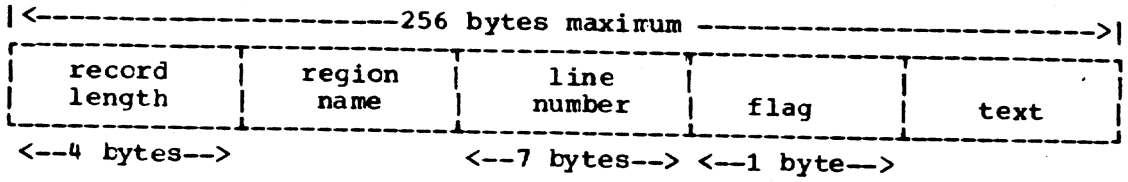
region
a line, or contiguous group of lines, whose numbers are prefixed by the same region name. The region of a data set is treated by the text editor as an entity; region names label a contiguous subset of lines for identification purposes. The length of the region name is determined by the value of the REGSIZE operand in the user profile.

The text editor automatically reorganizes the regions of a data set into alphabetically ascending order, by region name.

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

Region Name	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 500.
nextcase	0000100	Although NEXTCASE was the first region name
nextcase	0000200	entered, the text editor automatically
nextcase	0000300	alphabetizes regions.
swan	0000100	Initial allowable range of region name
swan	0000200	is from 0 to 244 characters.

region data set
is a VISAM data set that has this record format:



The record length field, which appears only in variable-length records, and the flag field are used only by the system. The region name, specified by the user, can be from 0 to 244 bytes long; its length is defined in the DCB suboperand of the DDEF command. Total record length, also defined in the DCB suboperand, cannot exceed 256 bytes. The beginning of a record in a region data set line is the first character of the region name. The relative key position is four.

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 EBCDIC characters are valid except a comma, equal sign, or semicolon that is external to all pairs of parentheses in the string. For example,

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

contains these normal strings:

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

A quoted string is any character string that is enclosed in apostrophes, within which all other apostrophes are doubled. All EBCDIC characters are valid. The representation of a quoted string after it is processed by the system does not have the apostrophes; doubled apostrophes are replaced by single apostrophes. For example:

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

transaction table
the text editor records the changes to a data set in this table; additions are noted in one set, deletions in another. The user can nullify the changes that are recorded in the transaction table (see "DISABLE, ENABLE, POST, and SET Commands," in Part III). Normally, when the editor is invoked, the transaction table is not active. To activate it, the user issues

```
default trantab=y
```

before invoking the editor. As long as TRANTAB=Y, the transaction table is active.

INVOKING THE TEXT EDITOR

The EDIT command invokes the text editor, initializes the transaction table if TRANTAB=Y, and invites the user to enter a command or a line of data. Unless preceded by a user-issued DDEF command, EDIT issues a DDEF that has these values:

DDNAME=EDDN, DSORG=VI or VP, DCB=(RKP=4, LRECL=132, RECFM=V, KEYLEN=XX)

The value of DSNAME is supplied by the user in the EDIT command; the value of KEYLEN is determined by EDIT by adding seven to the value of the REGSIZE operand that exists in the user profile. For example, if REGSIZE=2, then KEYLEN=9. The text editor can be used to create a data set. The two types of data sets that can be created are a region data set and a line data set.

Creating a Region Data Set

The system-supplied value for the REGSIZE operand is 0, the key length (KEYLEN) is 7, and a line data set is assumed. The user can change the value of the REGSIZE operand by specifying a number from 0 to 244. The REGSIZE operand governs the maximum length for the name of a region data set. The EDIT command can be used to specify both the value of REGSIZE and the name of a region in the region data set that is to be created or modified. If the user issues:

```
edit regds,rname=regnam,regsize=8
```

the system assumes that he wants to operate on the region named REGNAM in data set REGDS. The maximum length of region names in REGDS is 8 characters. The system then prompts (with a line number, if the LINENO operand is set at Y, the system-supplied value) for the first line in the region. The sequence of activity is as follows:

```
User:    edit regds,rname=regnam,regsize=8
System:  0000100
```

The system prompts for data for REGNAM, and the user enters the data. For example:

```
Sys,User:  0000100 this is data
Sys,User:  0000200 for region regnam
Sys:       0000300
```

The user wants to review these lines, but the system has printed the next line number. The user types in the break character (underscore) and a command, as follows:

```
Sys,User:  0000300 _list 0,last
System:    REGNAM 0000100 THIS IS DATA
           REGNAM 0000200 FOR REGION REGNAM
           _
```

The system displays the lines of the region and prompts for a command. Notice that the name of the region precedes each line number. In a region data set, the region name and the line number form the key for each line in the region. The user wants to terminate editing so he types the END command following the break character, as follows:

```
Sys,User:  _end
```

The user is now in command mode; he must reissue the EDIT command to continue processing his data set. Had the user not issued the LIST command at line 300, he could have terminated editing there:

```
Sys,User:  0000300_end
```

The break character is used when the system expects data (for example, it has issued a line number), and the user wants to enter a command. Had the user not entered the break character, the command at line 300 would have been put in the data set as data; it would not have been executed.

Note: Once a region data set has been created, the user cannot alter the maximum region name length (REGSIZE) for that data set. If he wants to allow region names of greater length, he must erase the data set and re-create it. Of course, while he is doing this, the user can hold the contents of the data set in a temporary data set.

Creating a Line Data Set

Unless the user changes the initial system value of REGSIZE, the EDIT command assumes a line data set; if LINENO=Y, the user receives a line number prompt:

```
User:      edit lineds
System:    0000100
```

Since LINEDS was a new data set, the system prompts with line number 100. The user can enter data, and the system responds with the next-highest line number, in increments of 100. To enter a command when the editor expects data, the user precedes the command with a break character:

```
Sys,User: 0000100  this is a line data set
          0000200  with only two lines
          0000300  _end
```

EDITING DATA SETS

The text editor can be used, also, to edit data sets and to create and edit data sets simultaneously. Whether the data set is line or region makes little difference in the procedure used. For the remainder of this discussion, a region data set is assumed.

The editing tools available to the user are extensive. The example that follows is by no means complete. The user should refer to individual command descriptions in Part III for more details.

Example: Previously, the user created REGDS as a region data set with one region, REGNAM. The value of REGSIZE was set to 8 for the data set. Now, whenever the user edits REGDS, the system expects him to treat the data set as a region data set with a maximum region name length of 8. The user can edit region REGNAM this way:

```
User:  edit regds,regnam
```

The region exists so the system responds by typing an _:

```
System:  _
```

Two lines of data that were created earlier exist in the region. Now, the user wants to create a new region called FIRSTONE:

```
Sys,User:  region firstone
```

The system prompts with line 100, since this is a new region. The user enters data into region FIRSTONE, and then he terminates the editing procedure with the END command.

```
Sys,User:  0000100 notice regsize is 8
           0000200 and lineno=y
           0000300 so far only two regions exist
           0000400 _end
```

When the user again edits this region he is prompted with an underscore. Assume now that the user wants to add two lines between lines 200 and 300 in increments of 25:

```
User:      edit regds,firstone
Sys,User:  _insert 200,25
           0000225  add these two lines
           0000250  between 200 and 300
           0000275
```

To continue adding lines in increments of 100 following line 400, the user issues another INSERT command, preceded by a break character:

```
Sys,User:  0000275  _insert 400
           0000400  more data can be entered
           0000500  and the system continues to prompt
           0000600  with line numbers in increments of 100
           0000700
```

To change line 300, the user types in, after the line number:

```
Sys,User:  0000700  _context 300,,only,just
System:    _
```

To check his results, the user enters:

```
Sys,User:  list 300
System:    FIRSTONE 0000300 SO FAR JUST TWO REGIONS EXIST
```

CONCATENATING INPUT RECORDS

The text editor accepts 256-byte records in a region data set. However, most input devices cannot deliver records of this length. Therefore, the input lines must be broken for entry at the input device and then concatenated by the system.

If the user wants to concatenate input lines, he must set the implicit operand, CONREC, to Y (the system-supplied default value is N). When

the user enters a line of data, and the last nonblank character of the line is the concatenation character (the system-supplied character is a colon), the system prompts with a colon and unlocks the keyboard so that the user can enter the rest of the line. The system joins the two lines to make one record; the first character of the second line replaces the concatenation character in the first line. For example:

```
User:       default conrec=y
Sys,User:   edit dataset1,rname=joe
            0000100 this line is :
            :  continued with the concatenation character
            0000200 _end
```

No line number was issued for the concatenation line; the line has become part of line 100. Also, a space was left between the last text character in line 100, and the concatenation character. If this space were not there, the two lines would be joined with no space between "is" and "continued." The content of line 100 is:

"this line is continued with the concatenation character"

The user can concatenate records for line data sets as well as for region data sets. He must remember, however, that the maximum record length for a line data set is 132 characters; for a region data set, it is 256 characters (including the region name).

For a line data set, concatenation works the same way. If the user does not want the colon as a prompt (as is done after line 100 in the example above), he sets CONPRMPT=N. For example:

```
User:       default conrec=y,conprmpt=n
Sys,User:   edit dataset2
            0000100 this line is :
            continued
            0000200 _list
System:     0000100 THIS LINE IS CONTINUED
            _end
```

The system unlocks the keyboard so that the user can enter the concatenation line.

Caution: If the concatenated record length exceeds 132 characters for a line data set, or 256 characters for a region data set, the 132nd or 256th character is replaced by a continuation character (the system-supplied character is a hyphen). The user receives a diagnostic message stating that truncation has occurred. The message contains the last five characters, entered before the record length was exceeded, and the continuation characters.

ENTERING HEXADECIMAL DATA

When using the text editor, the user can enter data in hexadecimal notation as well as in character notation. To enter data in hexadecimal notation, the user precedes the data with a letter X followed by a percent sign (X%). These characters, the system-supplied default for the HEXSW implicit operand, indicate that the data that follows is in hexadecimal notation. The user follows the X% with a string of hexadecimal data.

```
0000100 x%c1c2c3
```

The x% may occur anywhere in the data line:

```
0000100 defx%c1c2c3
```

All characters following the X% on the data line are treated as hexadecimal input until the end of the line or until a nonhexadecimal character is entered (hexadecimal characters are the numbers 0-9 and the letters A-F). When an uneven number of hexadecimal characters is entered, the system truncates the line to the last even character position:

```
0000100 x%c1c2c3c
```

is truncated to

```
0000100 x%c1c2c3
```

and a message is issued to inform the user of the truncation.

The processing is the same for the EDIT, REGION, INSERT, REVISE, UPDATE, LOCATE, and CONTEXT commands. See the LIST and CORRECT command descriptions, in Part III, for special considerations when using LIST and CORRECT.

Note: The x% symbol is never put into the data set; it merely tells the system that hexadecimal data follows.

Example: This example shows how to use text-editing commands with hexadecimal input; the LIST command, and the CHAR operand, are used to display the results of the operations.

```
User:       edit hexdata
Sys,User:   0000100 abcx%a1a2a3a4b1
            0000200 x%fafbfcfdgal       (hexadecimal input ceases
                                         with the first nonhexadecimal
                                         character, namely G)
            0000300 _list
System:     0000100 ABCstu              (unprintable hexadecimal
            0000200 GAL                  characters are ignored on
                                         output when CHAR is defaulted
                                         to C)

User:       list char=h
System:     0000100 C1C2C3A1A2A3A4B1    (If CHAR=H, all data is
            0000200 FAFBFCFDC7C1D3       printed in hexadecimal
                                         format)

User:       list char=m
System:     0000100 ABCA1stuB1          (If CHAR=M, all unprintable
            0000200 FAFBFCFDGAI          hexadecimal characters are
User:       insert 300                   printed as entered, and they
                                         are underscored)
System:     0000300 hex data fcllowsx%a1b2cffda5
            0000400 _list 300
            0000300 HEX DATA FCLLOWSv
User:       list 300,char=h
System:     0000300C8C5E740C4C1E3C140C6D6D3D3D6E6E2A1E2CFFDA5
User:       list 300,char=m
System:     0000300 HEX DATA FCLLOWSA1E2CFFDv
```

USING THE TEXT EDITOR

Text-editing commands can be issued only after EDIT or PROCDEF has been issued. Invoking the text editor does not, however, limit the user's access to the command system. He can issue any command while the editor is active.

Example: The user wants to write and compile a FORTRAN program. He enters these commands:

```
default lineno=n
edit source.ftnprog
```

To avoid confusion when entering FORTRAN statement numbers, the user has suppressed line number prompting. Following EDIT, his keyboard is unlocked. The input stream is as follows:

```
Sys,User: a=0.0
          b=17.9
          d=a*b
          write (2,5)d
          5 format (f10.6)
          stop
          end
```

The END statement in the last line is a FORTRAN statement. To terminate text editing, the user must issue the END command, preceded by a break character. To compile this program, he enters the compiler call, the name of the program, and Y to indicate that the program is cataloged:

```
ftn ftnprog,y
```

The program is compiled. He then enters, following the system prompt for a command statement, a direct call to execute his program:

```
Sys,User: ftnprog
System:   0.000000
          TERMINATED:  STOP
          -
```

Since the editor is still active, the user can change his source program with editing commands. He enters:

```
User:     revise 100,200
```

REVISE deletes the specified lines and requests replacement lines by unlocking the keyboard:

```
Sys,User: a=11.0
          b=14.0
          list 0,last
```

Since LINENO=N, the system responds to the LIST command by printing the lines without line numbers:

```
System:   A=11.0
          B=14.0
          D=A*B
          WRITE (2,5)D
          5 FORMAT (F10.6)
          STOP
          END
          -
```

Following execution of LIST, the system prompts for a command statement. The user can recompile and reexecute his program and can make additional alterations. To write another program, he issues another EDIT command. To terminate editing, he issues END.

DATA EDITING

The data-editing commands are used to build and edit VSAM and VISAM data sets. These commands are not as flexible as those of the text editor.

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

Table 6. Data-editing commands and their functions

Command	Function
DATA	Build a VSAM or a VISAM line data set.
LINE?	Obtain lines from a line data set or from a language processor listing data set. Print the lines on SYSOUT.
MODIFY	Insert, delete, or replace lines in VISAM data set or create a VISAM data set.

SOURCE INPUT

The system expects source input to follow certain commands, for example, DATA, EDIT, MODIFY, PLI, and PROCDEF. An operand, SYSINX, indicates whether the source input is expected from SYSIN or from a source list. If SYSINX=G (this is the system default value), the system expects to get input from SYSIN, which is either the terminal for a conversational task or the SYSIN data set for a nonconversational task. If you change this value to SYSINX=E, the system expects to get input from the source list; and if the source list is empty, the system goes to SYSIN for input. If SYSINX=L, the system goes to the source list only for input. (Example 3, below, shows how to use SYSINX=L. Refer to "Implicit Operands" in Section 6 and to Appendix C for the possible values of SYSINX.)

If the command (for example, EDIT) is executed from a PROCDEF, you may have created the expected data as part of the PROCDEF. The data immediately follows the command that expects the data. Then, to indicate that the input is to be taken from the source list, you must issue the command

```
DEFAULT SYSINX=E
```

before issuing the command that expects the data. If the source list is empty when SYSINX=E, the system goes to SYSIN for input. The examples shown below give some uses of SYSINX.

1. The following PROCDEF adds lines of data to an existing data set.

```
User:       procdef addata
Sys,User:   0000100 param DS=$1,$2,$3,$4,$5,$6
            0000200 default sysinx=e
            0000300 edit $1
            0000350 insert last
            0000400 $2
            0000500 $3
            0000600 $4
            0000700 $5
            0000800 $6
            0000900 __end
            0001000 default sysinx=g
            0001100 _end
```

When the PROCDEF is invoked (by entering ADDATA), the values entered for the parameters $2, $3, $4, $5, and $6 are treated as

input to the EDIT command. The values are added tc the data set named $1.

2. For a batch job, a user keypunches a card that contains a PROCDEF command for ADDATA and a series of other commands that are separated by semicolons. Other cards in the deck ccntain input for PROCDEF Y. Here, SYSINX=E.

```
User:  procdef addata;excise 0,last;end;procdef y
       param $1,$2,$3
       if '$1'='n'; set a=$2
       if '$1'=''; set a=$3
       _end
```

If SYSINX=G in this example, the second, third, and fourth cards are nct executed as input for the PROCDEF named Y. Rather, they are taken as input for ADDATA. The system attempts to execute the EXCISE and END commands after ADDATA has been created, but it cannot because there is no active processor. (A diagnostic is issued.)

3. If SYSINX=L, the system goes only to the scurce list for input. The fcllowing PROCDEF calls the text editor, and it contains a null line (line 400) as input to the editor.

```
User:       procdef nick
Sys,User:   0000100 default sysinx=1
            0000200 edit nickds
            0000300 line one of data
            0000400
            0000500 line three of data
            0000600 __end
            0000700 default sysinx=g
            0000800 _end
```

After PROCDEF NICK has been executed, the data set named NICKDS will contain a line (line 200) that is null.

If SYSINX=E, the system ignores line 400.

BULK OUTPUT

The bulk output commands (see Table 7) allow the user tc transfer data sets from his virtual storage tc output devices other than the terminal. The output printer, at the central computer installation, can write data sets more rapidly than the user's terminal. WT and PUNCH put data sets on tape and cards, which are not available at the user's terminal. Each bulk output command initiates a nonconversational task to accomplish the data transfer, thereby freeing the user from the need tc monitor bulk output.

Table 7. Bulk output commands and their functions

Command	Function
PRINT	Initiate printout of the specified data set cn high-speed printer.
PUNCH	Initiate transfer of the specified data set tc punched cards.
WT	Initiate writing of the specified data set cn magnetic tape, with tape in format for offline printing.

SECTION 3: PROGRAM MANAGEMENT

Language-processing and program control commands are used for program management.

LANGUAGE PROCESSING

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. PL/I and any processors supported by the PPLI are exceptions to this statement.

The user initiates source language processing by issuing the command for the desired language type. The language-processing commands are listed in Table 8. These commands are described in detail in Part III.

Table 8. Language-processing commands

Command	Language Type
ASM	Assembler language
FTN	FORTRAN language compiler
LNK	Linkage editor
PLI	PL/I language compiler
COBOL	COBOL Language Compiler (supported by PPLI)
FTNH	FORTRAN H Extended Language Compiler (supported by PPLI)
HASM	Assembler H Language (supported by PPLI)
PLIOPT	PL/I Optimizing Compiler (supported by PPLI)

A source program is a data set that contains source language statements. To be acceptable for language processing, a prestored source program must have line organization and must be named SOURCE.name. Source programs are automatically cataloged and retained by the system. For PL/I, and PPLI supported processors, any legal name is acceptable.

When source statements are submitted conversationally, or when they form part of the prestored SYSIN of a task, a source program is constructed with line organization. Each physical line entered into 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.

Note: The compilation of PL/I and PPLI supported programs is not interactive; you are not prompted for corrections to the source program. PL/I processing can only be initiated in a task for which the user issued a LOGON in 24-bit mode (that is, logon userid,,24).

STEPS IN LANGUAGE PROCESSING

From the user's standpoint, source-language processing proceeds in one of four ways. Items 3 and 4 below do not apply to PL/I or PPLI processing.

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 in the SYSIN data set. 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 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. Diagnostic messages for a single statement are written at the terminal, along with the incorrect line, and the user is invited to enter corrections. To indicate to the user that he can enter corrections, the system types a pound sign (#) at the beginning of a new line, and the keyboard is unlocked. The user may enter a correction line, the first part of which must be the line number that identifies the line being corrected, followed by a comma and the contents of the line. For example:

   Sys,User: #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 #:

   Sys,User: D,line number

   or he can enter:

   Sys,User: 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 processed by the language processor, and if no corrections are required for them, the next line is taken from the source program for processing.

   The user can enter other responses following the system's invitation (#) to enter corrections: I or C. If the user types I and presses the RETURN key, language processing continues without further display of diagnostics or invitations to enter corrections. This response is useful when the user determines that he has too many errors to correct conversationally. If the user types C and presses the RETURN key, he receives all diagnostics, but is not to be permitted to make corrections until language processing is completed.

4. The task is conversational, and the user enters his source statements from the terminal (that is, 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:

   Sys,User: 0000500 %line number,modification

The modification the user types in after the comma is his insertion or replacement line. He can delete a line or lines by typing after the system-issued line number:

Sys,User: 0000600 % D,line number

or he can enter:

Sys,User: 0000700 % D,first line number,last line number

The % indicates to the system that a modification follows. When the user enters the next source line that is not prefixed by %, the previously collected modifications are sent to the language processor, and the line is stored in the source program. This line is 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 restarts 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 is prompted with # to enter corrections. He can enter insertions, replacements, and deletions, as described for a conversational task with a prestored source program. (See Item 3, above.) He is prompted for corrections until he presses the RETURN key as the response to the # request. At that point, he is invited to enter his next source statement line.

The language processors display the incorrect line. However, when the incorrect line is part of a continuation line, only the last part of the line is displayed; this part may not contain the error. If the user wants to see the entire contents of the line, he:

1. Presses the ATTENTION key to interrupt source language processing;
2. Invokes the text editor and reviews the line in question;
3. Issues GO 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 conversational user if he wants to make modifications and restart or if he wants to continue processing.

When the user wants to continue, the next phase of the language processor is executed. If no errors are found that prevent the processor from producing an object program module, the user is informed.

Finally, the object module is stored in the user's library (USERLIB), unless he has defined another job library. If the object module is to be stored in another job library, 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 Assembler Programmer's Guide or FORTRAN Programmer's Guide. The linkage editor places the object module in the library specified in its input operands. For additional information, see Linkage Editor.

Express Mode: You can use express mode to compile or assemble more than one program or to request more than one linkage editor function without repeatedly entering the FTN, ASM, or LNK commands and operands. (PL/I uses the continue function to achieve the same result.) See the appropriate OS/VS language programmer's guide for similar options for program product languages supported by PPLI.

You set LPCXPRSS=Y in your user profile before invoking the language processor. If the task is nonconversational, the language processor reads the next record from SYSIN, following the completion of language processing. If the task is conversational, the system prompts you to enter the module name. The language processor assumes the same options for the next source program entered as those specified when the ASM, FTN, or LNK command was last specified. To terminate language processing, you can enter a break character followed by a command or you can interrupt the language processor when you are prompted to enter the module name. To initiate language processing again, enter the language-processing command and operands, but note that you are still in express mode (even though a system message said that express mode was terminated.)

You can interrupt the express mode by pressing the ATTENTION key. If you want to terminate express mode, set LPCXPRSS=N and issue a GO command, as follows:

```
default lpcxprss=n
go
```

The compilation, assembly, or linkage editing of the interrupted program is completed, and control is passed to the command system. However, when the processor encounters an error in the specification of a module name, a diagnostic message is issued, and language processing is terminated. Express mode is still active until LPCXPRSS=N.

Listing Data Sets

The user has complete control of the listings that are printed. The system action for the listing data set varies, depending on whether the symbol given as the module name has been previously used. 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) and maintains 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 adds the listing as a new generation to the existing generation data group. For example, the third listing data set for a given symbol becomes the latest generation (0); the second listing becomes the (-1) generation; and the first listing is erased.

The user can change the number of generations that are maintained in the generation data group associated with a given symbol. Assume he has been working with a module called MYPROG, and that he has two generations in his LIST.MYPROG generation data group. He can change the number of generations maintained in LIST.MYPROG.

Examples:

1. Catalog the two generations as separate data sets (for this example, MYPROG1 and MYPROG2).

```
catalog list.myprog(0),u,,myprog1
catalog list.myprog(-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 five generations, remove the oldest generations, and erase them.

```
catalog gdg=list.myprog,5,o,y
```

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 he issues:

```
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 on the PRINT command to remove one or more generations. (Refer to the descriptions of these commands in Part III.)

Programming Notes: The user can create source data sets and correct assembly or compilation errors with the text-editing commands. By leaving the text editor invoked while he assembles or compiles a program, the user can make changes after assembly or compilation is complete. Refer to Section 2, Text Editing, for a description of the text editor.

PROGRAM CONTROL

Program control system (PCS) 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 9.

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, AT or TRAP 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 USERLIB or from a job library (JOBLIB), all attributes are ignored. Private read/write storage is assigned to the CSECT, and the system does not recognize any of the above restrictions.

Caution: PCS commands are generally more difficult to use with PL/I programs or programs generated by PPLI supported compilers. Since no ISD is produced by any of these processors, symbolic names may not be used. In addition, actual hexadeci mal displacements from the module/procedure name are difficult to calculate since the name refers to the actual entry point of executable code. This entry point may be at various displacements in the object code although the listing might indicate displacement zero. However, once the correct instruction is located in the object code, an AT statement and most other PCS statements may be used. Complicated PCS statements should be avoided.

The user can employ PCS 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 (that is, true or false) conditions that allow or inhibit execution of other commands.
- Perform arithmetic computations.

Table 9. PCS commands and their functions

Command	Function
AT	Inform the user when execution of the program has reached a designated instruction location or make the statement that follows this command dynamic.
BRANCH	Dynamically change the control path of a program or resume execution at a different location.
CALL	Load and pass parameters to an object module and execute.
DISPLAY	Present the values of variables, the contents of machine registers, and the specified virtual storage locations to user's SYSOUT.
DUMP	Present the values of variables, the contents of machine registers, and the specified virtual storage locations to the task's PCSOUT data set.
GO	Resume execution of a previously interrupted program.
IF	Make the following statement conditional.
LOAD	Place an object module in the user's virtual storage without initiating execution.
QUALIFY	Allow the user to designate, before referring to group of internal symbols, the program in which the specified symbols are defined; thereafter, there is no need to explicitly qualify symbols.
REMOVE	Selectively delete the previously entered dynamic statements (that is, those that include AT).
RUN	Initiate execution of the loaded object module; resume execution of the interrupted program; load and initiate execution of the object module. (Restrictions on the use of RUN are given in its command description in Part III.)
SET	Change the contents of machine registers, the values of program variables, the virtual storage locations, or the command symbols.
STOP	Interrupt execution of the user's program; display the instruction location or the FORTRAN statement number where interruption was handled (if LIMEN=I).
TRAP	Requests notification when execution of an object program causes certain events to occur.
UNLOAD	Remove the specified object module from the user's virtual storage.

USE OF COMMAND STATEMENTS

PCS commands are often conveniently expressed in command statements. For purposes of this discussion, three types of command statements are considered: dynamic, immediate, and conditional.

DYNAMIC STATEMENT: This is a command statement that contains an AT or TRAP; the AT or TRAP should appear first in the statement and should be the only AT or TRAP command in the statement. Commands that precede AT or TRAP are executed immediately; commands that follow AT or TRAP are not executed until control arrives at the instruction location designated by the AT command, or the TRAP event occurs. Only these commands can follow AT or TRAP 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. A dynamic statement has the form,

```
AT location;command
TRAP operands;command
```

IMMEDIATE STATEMENT: This is a command statement that does not begin with 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, the commands making up the statement are executed from left to right until an IF that specifies an unsatisfied condition is encountered, or the end of the command statement is reached. Any command may appear in a conditional statement. A conditional statement has the form,

```
IF condition;command
```

PCS APPLICATIONS

To load an object module, the user can issue a LOAD, CALL, or QUALIFY command or he can issue a direct call. (Refer to the descriptions of these commands in Part III.) The loading of one module may cause another module that is implicitly referenced by the first module to be loaded. For 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 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, with the QUALIFY command, 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.

If the user has previously identified an external name via the QUALIFY command, he can refer implicitly to locations in the identified module. He simply omits the external name from the PCS command operand. For example, to display the location four bytes from CSECT PGM, the user can enter:

```
display pgm.(4)
```

or he can enter:

```
qualify pgm
display .(4)
```

The effects of the QUALIFY command last until the user issues another QUALIFY command or until he unloads the module. So, later in his task, he can still use the implicit form for addressing. This implicit form applies to all PCS commands.

Note: Any PCS command causes a module to be loaded if the user enters an external name as the operand of the PCS command. (SET PGM. (4)='A' causes PGM to be loaded).

To display a location offset from zero, the user can enter:

```
display L'0'.(4)
```

or he can enter:

```
display .(4)
```

if no external name has been identified by a QUALIFY command.

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 causes PGMB to be unloaded if PGMA is the only loaded module that references PGMB.

Note: If PGMA is referenced in any PCS command, unloading PGMA removes all dynamic statements that refers to it during the session. If PGMA is not referenced in a PCS command, but PGMB is, all dynamic statements referencing PGMB 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 OPERAND SPECIFICATION

The user has broad addressing capabilities for referencing his programs by using variables, constants, and the dynamic statement counter as

operands for PCS commands.

VARIABLES: These are designated by their (1) symbolic names, (2) hexadecimal locations, or (3) register numbers.

1. Symbolic names: PCS commands use either external, internal, or command symbols.

   - External symbols are defined within a program for reference during load or execution. FORTRAN COMMON BLOCK names, function names, subroutine names, and the names of assembler language ENTRY and CSECT statements are external symbols. For example, an assembler program named PGM has these characteristics:

     Two control sections, named PGMCS and PGMPS.

     Two ENTRY statements, named PGMEP and PGMEX. These, then, are valid external symbols:

     ```
     PGM
     PGMCS
     PGMPS
     PGMEP
     PGMEX
     ```

     Four external symbols are assigned to every FORTRAN object module

     ```
     module name            (for example, FTNPGM)
     CSECT name             (for example, FTNPGM#C)
     PSECT name             (for example, FTNPGM#P)
     module entry point     (for example, FTNPGM#E)
     ```

     In PCS 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.

     The user may refer to internal symbols only if he requested an 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 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), followed by a period; the original name of the defining program (second level), followed by a period; 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 referenced by entering ABX alone.

Note: If a program processed by the linkage editor contains an internal symbol that is 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 nor 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 PCS commands.

  When a command symbol has been defined, it is addressable for the user's entire terminal session; it is not affected by unloading one of his programs. The command symbol may be retained for future terminal sessions by using the PRCFILE command (see Section 6).

  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.

  The types of internal and external symbols are discussed below.

- %CSECT and %COM are two special symbols that may be used to refer to the unnamed assembler language control section and the FORTRAN COMMON BLOCK, respectively. %CSECT may be used only as an internal symbol; %COM may be used as either an internal or external symbol. When multiple unnamed control sections are loaded internal symbols (including %CSECT) are referenced in the last-loaded control section even when the internal symbol is explicitly qualified by the module name.

- 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 I/O statements. Nonexecutable statements are specification statements and subprogram statements; they should not be referenced.

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 referenced because it is not executable.

A statement number refers to a statement's first line and all of its continuation lines; continuation lines can not be designated separately when using incremented statement numbers.

The integer 0 may be used to refer to a program's first executable statement when the first executable statement is not numbered. 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 within an array. A subscript to an internal symbol must be either an integer constant, an integer variable, or an integer arithmetic expression.

  Symbols used in subscripts may contain subscripts, and subscript symbols may contain offsets; however, the subscript value obtained when the various nests and expressions are evaluated must be an integer. Five levels of nesting (subscript and subscript, subscript and offset, or offset and offset) are allowed. The user may not refer to dummy FORTRAN arrays in a PCS statement. If it is necessary to refer to a shared array, references to the array should be qualified by the module that contains the array, that is, the main routine.

  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 (1) is not an integer greater than 0, or (2) is larger than the dimensions defined for the array.

  Examples:

  a. 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.

b. 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, the result is,

```
ARRAY (2,3) = 15
```

c. 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 that is 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 the values in the expression,

```
ARRAY (ARRAY (3,2),ARRAY (1,4))
```

reduces the expression to one similar to Example b. The final value is determined by substituting the values from ARRAY in Example a:

```
ARRAY (1,5) = 13
```

d. The subscripted symbol to be evaluated is:

```
ARRAY (1+X/Z,X-Y*Y)
```

Assume that X=6, Y=2, and Z=4. The arithmetic expressions must be evaluated first.

```
1+X/Z = 1+6/4 = 1 + 1 = 2
```

(Note that in integer division PCS ignores the remainder.)

```
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. 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.

- Offset, length, and type reference a specific byte following a symbolic address or hexadecimal 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 or address and its offset. The form is symbol (or address), period, left parenthesis, offset, comma, length in bytes, comma, type of output format, right parenthesis:

```
SYMBOL    .(OFFSET,LENGTH,TYPE)
ADDRESS
```

An offset may be one of the following:

integer, hexadecimal, or address constant
integer or hexadecimal variable
integer or hexadecimal arithmetic expression

Length must be a positive integer.

TYPE controls the output format; it is specified as:

C -- character format; a string of characters is displayed with all unprintable characters represented by periods.
I -- integer format; a string of from 1 to 10 integers preceded by a sign (for example, +1234567890).
B -- binary format; a series of binary digits in bit representation. The LENGTH attribute for a binary constant contains the length, in bytes. (In a SET command, this is specified in the form B'11001010', where each digit represents a bit.)
F -- floating-point format; for single precision values, this specifies 8 digits in floating point format (that is, ±.XXXXXXXXE±XX). For double precision, this specifies 16 digits in double precision format (that is, ±.XXXXXXXXXXXXXXXXD±XX).
S -- symbolic assembler language format; as output data, this is a header and one or more lines of assembler language code:

```
    LOC   INSTRUCTION  LABEL  OPC  OPERAND     SYMBOL
01 00022   4330 F042    NEXT   IC  3,66(0,15)  SWITCH
```

If the user does not specify the output format, the data is displayed in hexadecimal format. If the user specifies a range and uses two different output formats in the statement (for example, PGM.(4,,C):PGM.(7,,I),) the last form specified is used.

Note: The module must have an ISD if the user wants to specify TYPE=S.

In a SET command, the TYPE attribute is ignored; the data is entered as it is specified on the right side of the equal sign (for example, SET PGM.(4,,I)='ABCD') results in character data

being entered; the I, on the left side of the equal sign, is ignored.

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:

a. The twenty-seventh byte beyond DATA would be expressed as:

```
data.(27)
```

or it would be expressed as:

```
data.(x'1b')
```

If a length of four bytes is to be attributed to the data at the twenty-seventh byte from DATA, the expression would be:

```
data.(27,4)
```

or it would be:

```
data .(X'1b',4)
```

b. The user may 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)
```

c. A four-byte field that is the twentieth fullword field in a table whose address is A'DATA' would be expressed as:

```
.(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 and no qualification exists.

d. It is possible to achieve a full virtual storage dump by specifying the range from location 0 to FFFFFF (for 24-bit addressing) and 0 to FFFFFFFF (for 32-bit addressing) as offsets in the operand field of the DUMP command, for example:

```
dump .(x'0'):.(x'ffffff')
```

2. Hexadecimal locations: PCS commands can refer to the contents of locations. The hexadecimal address of the location referred to is enclosed in apostrophes and is 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'
```

   Note that a hexadecimal address can be used in place of a symbol for use with offset.

```
L'0'.(X'800',6)
L'1AF000'.(X'24',X'18')
L'1AF000'.(,24)
```

3. Registers: PCS commands can refer to any of the general or floating-point registers. A reference to a general register is written as nR, where n is an integer from 0 to 15 that identifies the register.

   A reference to a single-precision, floating-point register is written as nE, where n is 0, 2, 4, or 6. A double-precision, floating-point register is referenced by writing nD, where n is 0, 2, 4, or 6.

   Examples of register references:

```
3R represents general register 3
2E represents floating-point register 2, single-precision
6D represents floating-point register 6, double-precision
```

CONSTANTS: Six classes of constant are used in PCS commands: (1) integer, (2) character, (3) hexadecimal, (4) floating-point, (5) address, and (6) binary.

1. Integer constant may be written as a signed or unsigned 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 is in storage. If the length of the constant is not appropriate in the context used, the constant is truncated or is 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 zeros 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 or end in any position within the decimal number, as appropriate.

   An exponent specifies a power of 10 by which the principal part is 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 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 -75 to +75. If the exponent exceeds the maximum, +75 is assumed. If it exceeds the minimum, 0 will be assumed.

   A floating-point constant is 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 are 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, and 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)'
```

6. Binary constant is written as a string of binary digits enclosed in apostrophes and preceded by the letter B. Eight bits comprise one byte of data. When the user displays binary data, the data is displayed as a series of bytes.

   Examples of binary constants:

```
B'10001100'
B'01' (displayed as B'00000001')
B'10' (displayed as B'00000010')
B'1010' (displayed as B'00001010')
```

   When the user enters a binary constant on the right side of the equal sign in a SET command, and if LIMEN=I, the SET operand is displayed for review in hexadecimal notation. For example, if the user enters

```
default limen=i
set pgm.(4,2)=b'1111000011111111'
```

   the system displays

```
PGM.(X'4',2)=F0FF
```

   to confirm the operation.

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 % when the AT or TRAP command is entered. (For example, AT X;DISPLAY%.) The value of the counter may be displayed or dumped, and can be used in forming expressions. The counter that is displayed or dumped is the one associated with the AT or TRAP statement in which the counter is referenced. Since % is not a user's variable, it cannot be changed by a SET command. (See "Examples Using PCS Commands," Example 3, later in this section.)

OPERAND DEFINITIONS

The terms used to describe the operands of PCS commands are (1) data location, (2) data field, (3) expressions (arithmetic and logical), (4) instruction location, (5) link-edited module name, (6) object module name, and (7) statement number.

DATA LOCATION: A symbol, a hexadecimal location, a register, or the special counter (%) can be a data location. Both fully defined 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 attribute or the length attribute. A length attribute can be assigned to a symbol with offset. The offset follows the period and left parenthesis and is followed by a comma and a length. Length is specified as an integer or

a hexadecimal constant that is greater than 0. The attribute is closed with a 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 a 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'
```

DATA FIELD: A contiguous group of storage locations whose contents are to be displayed or dumped is a data field. 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 CSECT name, written as an internal symbol without an offset, specifies the entire CSECT as a data field; so does a CSECT 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. The register types are:

```
R = general registers
E = floating-point registers with fullword form
D = floating-point registers with doubleword form
```

When the user specifies a range, general register 0 follows general register 15, just as floating-point register 0 follows floating-point register 6.

Examples of specifying a range of registers are as follows:

```
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 fullword format

6:2D     floating-point registers 6, 0, and 2 in doubleword format
```

A data field may be specified by a symbolic range that 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 is 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 CSECT. 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 are:

```
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)
```

EXPRESSIONS: PCS command expressions are either arithmetic or logical. They are formed by using these operators:

Operator	Meaning
Arithmetic	
+	Addition
-	Subtraction
*	Multiplication
/	Division
Logical	
¬	Logical NOT
&	Logical AND
|	Logical OR
Relational	
>	Greater than
<	Less than
=	Equal to
>=	Greater than or equal to
<=	Less than or equal to
¬=	Not equal to
¬>	Not greater than
¬<	Not less than

1. Arithmetic expressions may be used as subscripts or offsets, as values to which variables are to be set, as values to be compared when you use relational operators, 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 in a dynamic statement to reference the dynamic statement counter.

   These rules must be followed in the formation of arithmetic expressions:

   a. Any arithmetic expression may be enclosed in parentheses.

   b. 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.

   c. An arithmetic element or expression preceded by a sign (+ or -) is permitted; the operators * and / must be preceded and followed by elements or expressions or both.

d. All data locations connected by arithmetic operators must have lengths of 256 bytes or less and be aligned on the appropriate boundary.

Arithmetic expressions that do not contain terms that are in parentheses 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 terms in parentheses are evaluated by treating the innermost term in parentheses first. After all terms in parentheses have been evaluated, the remaining operations are performed in the same way as is done for expressions not in parentheses. 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. Therefore, if 13 is divided by 2, the answer is 6 (13/2=6). The expression A*B/C may yield a different result than the expression B/C*A.

For example, where A=8, B=6, and C=4, the first expression is

```
8*6/4=12
```

and the second expression is

```
6/4*8=8
```

Examples of valid arithmetic expressions:

```
1.E-5
PGM.X.(4)
PGM.X/PGM.Y-1
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 (for example, 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.

2. Logical expressions are used in a conditional statement and take any of these forms:

   a. A single logical variable.

   b. Two or more logical variables connected by the logical operators & or |.

   c. Two arithmetic expressions of the same type, connected by a relational operator.

   A logical expression that contains a relational operator has the logic value "true" if the condition expressed by the operator is met when the expression is evaluated. Otherwise, the expression has 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 terms in parentheses are evaluated in the following order:

   a. Multiplication and division (* and /)

   b. Addition and subtraction (+ and -)

   c. Relational operators (>, <, =, >=, <=, ¬=, ¬<, ¬>)

   d. Logical NOT (¬)

   e. Logical AND (&)

   f. Logical OR (|)

   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 is 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 is 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.

Terms in parentheses within logical expressions are evaluated in the same order. Then, when the expressions have been reduced (that is, a single logical value has been assigned to each term in parentheses), evaluation is 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:

a. The relational operators =, >, and <.

b. A single logical variable, in which case the variable need not be enclosed in parentheses.

c. A compound logical expression, in which case the expression must be enclosed in parentheses.

Assuming that 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. Thus, if A is false, and B is true, then ¬A|B is true, and ¬(A|B) is false.

INSTRUCTION LOCATION: A statement within the user's source program is an instruction location. An instruction location is expressed either as the statement number of an executable FORTRAN statement or as an internal symbol in a source program that is written in assembler language. In either case, the user can apply an offset to the primary location designator. An explicit length is ignored. When an internal symbol is used, it does not have to reference a location defined in the ISD as an instruction or as a CSECT 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 that is produced may not contain all of the in-

formation 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.

LINK-EDITED MODULE NAME: This name must precede the original program name, when qualifying internal symbols in a program that has been processed by the linkage editor.

OBJECT MODULE NAME: This name is always the one assigned when the source module was compiled or assembled. When internal symbols are referenced, the object module name must qualify the symbol. This name must be further qualified if the original program module was processed by the linkage editor.

STATEMENT NUMBER: This 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 PCS command names and operands may be used. Examples of valid synonyms are:

```
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 are 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 are substituted for LEPGM, but are not substituted for PGM or IOSR. (Refer to Section 6 and to the description of the SYNONYM command in Part III.)

Examples:

1. Assume the user has link-edited programs PGMA, PGMB, and PGMC that form a new program, LEPGM. Now, the user wants to reference concurrently internal symbols within PGMA, PGMB, and PGMC with PCS commands. Since only one qualification is allowed at one time, the user is required to qualify fully all symbols in two of the three program modules involved.

   Suppose he enters:

   ```
   synonym a=lepgm.pgmb
   synonym b=lepgm.pgmc
   qualify lepgm.pgma
   ```

   Now explicit qualification is 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 be required otherwise. 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 enters the definitions:

```
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, instead, displays an element of the table.

Note: Substitution is made for ARRAY, I, and J, since each is a data location. If ARRAY had been explicitly qualified (PGM.ARRAY), then TABLE. would not have been substituted, since ARRAY is the second character string in the data location PGM.ARRAY.

Examples Using PCS Commands

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 PCS reaches the ERREXT location

```
at errext;display 0:15r,0:6D;dump top:bot
```

2. The user wants to change the value of variable PCINT 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 the CSECT named BLDTAB:

```
at entab;if %=(%/10)*10;display tab;if %=(%/100)*100;dump bldtab
```

4. The user wants to use PCS commands to produce input and output to his program. He wants to make some computations, using the sequen-

tial 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 has 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 PCS commands, he can fix his program temporarily, without reassembling, by issuing

```
at csect.(x'176');branch csect.(x'124')
```

6. The user wants to display the contents of all general registers when the variable VAR1 in his PSECT changes.

```
TRAP STORE,VAR1;DISPLAY 0:15R
```

SECTION 4: COMMAND CREATION

The user can alter the names of system-supplied commands, redefine system-supplied commands, and create new commands from a series of system-supplied commands, or user assembler object coding. In creating a new command, the user can also define his own operands and establish the desired defaults for these operands.

Two system-supplied commands, PROCDEF and BUILTIN, are used to create new commands.

- 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.

The following are some advantages to user-written commands:

1. Although it is possible to store a series of commands (for example, a nonconversational 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 create 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 task or several tasks. He can collect these commands in sequence as a procedure, assign a name to this procedure via PROCDEF, and invoke this procedure 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 task, just as any system-supplied command is entered. Defining a command procedure is analogous to writing a computer program; that is, a set of commands is established at one time and is executed at a later time by issuing its name.

2. The user may require an entirely new command that 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 sequence of command statements that uses parameters 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 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, which are commands, are scanned and executed in the same manner as a system-supplied command that is entered at the terminal.

PROCEDURE LIBRARY

Command procedures are stored in procedure libraries: user-written procedures are in the user's procedure library (member SYSPRO of USERLIB); system-defined procedures are in the system procedure library (member SYSPRO of SYSLIB). When the user issues a command, the system first searches the user's procedure library. If the procedure is not there, the system checks its library. This order of library search enables the user to name a procedure he has created with a system-supplied command name, thereby preempting execution of the system procedure.

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. (See the description of the command in Part III.)

When the user enters the PROCDEF command, the text editor is invoked. The user can use all of the text-editing commands during command creation (see "Editing Procedures" later in this section and "Text Editing" in Section 2 of this part). Unless the user suppresses line number prompting (by issuing the command DEFAULT LINENO=N), the system prompts him to enter data by issuing line numbers. For a new procedure, the system issues line number 100; for an existing procedure, the system issues an underscore as a prompt for a command. In both cases CLP is set to the first line after line 0.

For example, if COPYCAT is the name of the command procedure being defined, the user enters

```
procdef copycat
```

and the system replies

```
0000100
```

If COPYCAT were previously defined, and the last line is line 800, the system's reply is an underscore and CLP is set to the first line of the PROCDEF.

SPECIFYING DUMMY OPERANDS

The user can build a command procedure that accepts operands when called by its command name; to do this, he must establish dummy operands in the PROCDEF to "hold the places" for the real operand values. Dummy operands are placed in the PROCDEF where the real values would be expected; then, when the command is entered, the real operand values are substituted for the dummy values. The dummy operands are specified in a PARAM line. This line must only occur at the first line after line 0 of a PROCDEF; the format is:

```
User:       procdef inlaw
Sys,User:   0000100 param trouble
```

Then, when the PROCDEF is called, the system accepts one operand (for example, INLAW JONES) and substitutes the operand for each occurrence of the dummy operand in the PROCDEF.

```
User:       procdef inlaw
Sys,User:   0000100 param trouble
```

```
0000200 display 'trouble'
0000300 _end
```

In this example, the PROCDEF could be called like this:

```
inlaw jones
```

The system substitutes the operand JCNES for each occurrence of the dummy operand TROUBLE. The PROCDEF displays the word JCNES.

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 replaces all occurrences of the dummy operand in the procedure text. For 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 replaces the occurrences of DSNAME in the procedure text.

2. An external character 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) is replaced by a substitute in the procedure text when the procedure is called. For 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 replaces $1 in the procedure text. Operand resolution is discussed in detail under "Operand Resolution and Substitution" later in this section.

   Example:
   Procedure

   ```
   User:       procdef callme
   Sys,User:   0000100 param entry=$a
               0000200 display 'calling $a'
               0000300 call $a
   ```

   Procedure call can be in one of the following forms:

   ```
   callme entry=one
   callme one
   ```

   The result is:

   ```
   CALLING ONE
   ```

   Module CNE is called.

   ENTRY in the PARAM line is an external string and is associated with the keyword ENTRY in the procedure call. The value ONE is substituted for the dummy operand $A in each occurrence.

Procedure

```
User:       procdef callme
Sys,User:   0000100 param $a
            0000200 display 'calling $a'
            0000300 call $a
```

Procedure call can be in one of the following forms:

```
callme one
callme $a=one
```

The result is:

```
CALLING ONE

Module ONE is called.
```

$A in the PARAM line is both the external string, used for association with the calling operands, and the internal string, to be substituted for in the procedure.

The user may specify a dummy operand as either a normal string or a quoted string. (Refer to the definition of string constants in Section 2 under "General Terms.")

Note: Dummy operand values are usually preceded by a $ or some other identifier to ensure that only desirable substitution occurs. (See "Operand Substitution," later in this section.)

ENTERING PROCEDURE TEXT

After the user issues the PROCDEF command name and operands, and optionally the PARAM line, all subsequent lines issued without a preceding break character are included in the procedure text. The system prompts for each line with a line number, and there is no limit to the number of lines the user can enter.

When a break character appears first in a line, the statement that follows is interpreted as a command. (See the list of definitions under "General Terms" in Section 2.) However, when the first and second characters of the line are break characters, the usual break-character action does not occur. Instead, the system replaces the pair of break characters with a single break character and processes the line as if it were text. Thus, lines starting with break characters can be put into procedures.

The user can enter system-supplied commands (including PROCDEF and BUILTIN) or user-written commands as text. 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 can remain variable, if the user specifies dummy names that also appear in the PARAM line, or can be fixed with explicit values. Fixed operand values are not included in the PARAM line and are acted upon as specified in the text when the procedure is called.

A direct call to an object module can be entered by using the name of the module in the procedure text.

TERMINATING PROCEDURE DEFINITION

The user terminates processing of a PROCDEF command by entering a break character followed by either an END command, another PROCDEF command, a

PLI command, or an EDIT command. When the user enters another PROCDEF command, the same options for terminating its processing are applicable; the last PROCDEF must be terminated with either an END, EDIT, or PLI command.

Examples:

1.
```
User:       procdef copycat
Sys,User:   0000100 param ddname=alphname,dsname=name1,-
            0000200 $any,newname=$1
            0000300 ddef ddname=alphname,dsorg=vi,dsname=name1,-
            0000400 volume=$any
            0000500 catalog dsname=name1,u,newname=$1
            0000600 _end
```

In line 100, DDNAME, DSNAME, and NEWNAME are external strings (keywords) that associate the calling parameters with the internal strings in the line: ALPHNAME, NAME1, and $1, respectively. These internal strings are replaced in the procedure text by the calling parameter values. $ANY, represented positionally in line 100, is replaced by a substitute in the text.

DDEF, on line 200, is a system-supplied command that has the variable operands DDNAME, DSORG, DSNAME, and VOLUME. DSORG=VI is a fixed operand value and is acted upon as specified. Values for the other variable operands are supplied when the procedure is called.

CATALOG, on line 300, is also a system-supplied command. Two of its operands are variable and are replaced by substitutes when the procedure is called. When the break character, followed by the END command, is entered, the definition of this procedure is terminated.

2.
```
User:       procdef dmprog
Sys,User:   0000100 param $1, 'here:there'
            0000200 if '$1'='yes'; display 'success'
            0000300 if '$1'¬='yes'; dump here:there
            0000400 _edit xyz
```

The quoted string 'HERE:THERE' is given as a dummy operand in the PARAM line. The apostrophes permit the use of the special character (the colon) within the character string. The apostrophes are 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 are not removed when the substitution is effected. The break character and EDIT command on line 400 terminate the PROCDEF.

3.
```
User:       procdef diff
Sys,User:   0000100 param alphname, $1, name1
            0000200 ddef ddname=alphname,dsorg=$1,dsname=name1
            0000300 _procdef callme
            0000100 param alphname,$a, $1, $2, '$3', $4
            0000200 asm name=alphname, macrolib=$a,y,y,y,y
            0000300 copycat alphname, $1,$2,$3,$4
            0000400 _end
```

The break character preceding the second PROCDEF command terminates the execution of PROCDEF DIFF. The PROCDEF named CALLME is terminated with _END. Within CALLME (line 300), the user-written command COPYCAT is used, since it was defined by a PROCDEF.

Nested PROCDEFs

The text of a procedure, defined by PROCDEF, may contain other PROCDEF commands. This structure is called a nested PROCDEF.

SYSINX: The value of SYSINX determines the source of input for a PROCDEF command. A nested PROCDEF should get its input from the preceding PROCDEF. (See Example 1.) Before entering the nested PROCDEF, issue the following command:

```
default sysinx=e
```

The user must remember to return the value of SYSINX to G following completion of the nested PROCDEF. (For further information on SYSINX, see the discussions under "Source Input" in Section 2 and "Implicit Operands" in Section 6. The system default is given in Appendix C.)

Examples:

1.
```
User:       procdef abc
Sys,User:   0000100 param @1,alphname,dsname=name1,newname=$n
            0000200 ddef alphname,vi,name1
            0000300 catalog dsname=name1,newname=$n
            0000400 default sysinx=e
            0000500 procdef @1
            0000600 param @2,name2,dsname=name3
            0000700 ddef name2,vs,dsname=name3
            0000800 set r=5
            0000900 qualify @2
            0001000 @2
            0001100 __end
            0001200 default sysinx=g
            0001300 _end
```

Lines 100 through 1200 are treated as text of ABC. This procedure is terminated by _END, in line 1300. The two break characters that precede END in line 1100 are replaced by a single break character, the normal break-character action does not occur.

When ABC is entered, the values of the calling parameters are resolved and substituted for the occurrences of the dummy operands throughout the procedure. ABC is executed so that DDEF, CATALOG, and DEFAULT are executed. When the PROCDEF on line 500 is encountered, definition of a new procedure is initiated. If the command ABC had been issued as

```
abc one,myprog,dsname=mylib,mylib02
```

line 500 would become

```
PROCDEF ONE
```

Input for this PROCDEF comes from within ABC since SYSINX=E (line 400). Lines 600 through 1000 of ABC become lines 100 through 500 of ONE and the END on line 600 terminates procedure definition.

```
PROCDEF ONE
0000100 PARAM @2,NAME2,DSNAME=NAME3
0000200 DDEF NAME2,VS,DSNAME=NAME3
0000300 SET R=5
0000400 QUALIFY @2
0000500 @2
0000600 _END
```

The next time the user issues ABC, with the first operand specified as anything but ONE, another new procedure is defined. If ONE is specified again as the first operand of ABC, no new lines are added to PROCDEF ONE. In order to add lines to a procedure created on line 500 of ABC, there must be an INSERT LAST command in ABC.

Notice that when ABC is executed, the value of SYSINX is returned to G (line 1200).

2.
```
User:       procdef def
Sys,User:   0000100 param $1,$2,alphname,dsname=$3
            0000200 ddef alphname,vi,dsname=$3
            0000300 default sysinx=e
            0000400 procdef $1
            0000500 param dsname=$3,$4,$2
            0000600 catalog dsname=$3,state=u,newname=$4
            0000700 default sysinx=e
            0000800 procdef $2
            0000900 param @a,@b,@c
            0001000 if '@a'='yes'; display @b
            0001100 if '@b'='yes'; display @c
            0001200 ___end
            0001300 __end
            0001400 default sysinx=g
            0001500 _end
```

In procedure DEF, there are two nested PROCDEFs. The PROCDEF on line 800 is nested within the PROCDEF on line 400, which is nested within DEF. The first time DEF is executed, a procedure is defined (via PROCDEF on line 400); when this procedure is executed, another procedure results.

Lines 100 through 1400 are text of DEF. The first two break characters preceding END on 1200 and the two break characters preceding END on 1300 are replaced by single break characters. Assume DEF is called:

```
def two,three,mypro,dsname=myjob
```

Resolution and substitution of calling parameters occurs; DDEF and DEFAULT are executed, and then PROCDEF on line 400 is executed. This procedure definition results:

```
PROCDEF TWO
0000100 PARAM DSNAME=MYJOB,$4,THREE
0000500 CATALOG DSNAME=MYJOB,STATE=U,NEWNAME=$4
0000300 DEFAULT SYSINX=E
0000400 PROCDEF THREE
0000500 PARAM @A, @B, @C
0000600 IF '@A'='YES'; DISPLAY @B
0000700 IF '@B'='YES'; DISPLAY @C
0000800 __END
0000900 _END
```

Now, assume this procedure call occurs:

```
two dsname=mine,ok,three
```

Calling parameters are resolved and substituted, CATALOG and DEFAULT are executed, and another procedure is defined:

```
PROCDEF THREE
0000100 PARAM @A,@B,@C
0000200 IF '@A'='YES'; DISPLAY @B
```

```
0000300 IF '@B'='YES'; DISPLAY @C
0000400 _END
```

Following these procedure calls, the user has three procedures: DEF, TWO, and THREE.

3.
```
User:       procdef job
Sys,User:   0000100 param a>1,@2,alphname=name1,$3
            0000200 ddef name1,vi,$3
            0000300 default sysinx=e
            0000400 procdef a>1
            0000500 param $3,newname=$4
            0000600 catalog $3,state=u,newname=$4
            0000700 __procdef @2
            0000800 param a1,b2,c3
            0000900 if 'a1'='b2';display c3
            0001000 __end
            0001100 default sysinx=g
            0001200 _end
```

Within JOB, there are two nested PROCDEFs, but this time they are both nested within JOB, not one within the other, as in Example 2. When JOB is called, two procedures are defined and stored in the procedure library. Assume this procedure call:

```
job cat,cond,alphname=ddx,datas
```

Following resolution and substitution of the calling parameters, DDEF and DEFAULT are executed, and the PROCDEF on line 400 is encountered. This procedure is then defined:

```
PROCDEF CAT
0000100 PARAM DATAS,NEWNAME=$4
0000200 CATALOG DATAS,STATE=U,$4
```

Procedure definition is terminated by the PROCDEF (preceded by a break character) on line 700 of JOB. A new procedure is then defined:

```
PROCDEF COND
0000100 PARAM A1,B2,C3
0000200 IF 'A1'='B2'; DISPLAY C3
0000300 _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:

```
User:       procdef tab
Sys,User:   0000100 param ddname=arg1,dsrg,dsn1,$new$,@c
            0000200 ddef arg1,dsrg,dsn1
            0000300 mycat $new$
            0000400 mycall @c
            0000500 _end
```

When TAB is called, DDEF is executed and MYCAT is recognized as a user-written command. MYCAT invokes its procedure, and if that procedure calls another procedure, that call is processed. When execution of MYCAT is completed, MYCALL, also a user-written command, is executed.

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 (that is, 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, the user enters a break character followed by the EXCERPT command. He specifies as operands on the EXCERPT command the name by which he refers to the owner's user library (in the above example, LIB), the member name SYSPRO, and the name of the owner's procedure. The entire text of the procedure is inserted into the sharer's user library.

For example, a user wants to share a command, procname, 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: 0000100_excerpt lib(syspro),rname=procname,100,last
          _end
```

ABC is defined as a command in the sharer's user library and may be called by him.

EDITING PROCEDURES

The PROCDEF command invokes the text editor. This enables the user to issue any of the text-editing commands while he is defining a procedure or after he has defined it. The user enters a break character, followed by the text-editing command. He should not enter the EDIT command.

The CORRECT command can be used within a line to respecify characters that the user wants to insert, replace, or delete. Other commands, such as INSERT, EXCISE, and REVISE, can be used to insert, delete, or replace complete lines in the procedure text. (See Part III for the descriptions of these commands.)

This example shows how the text-editing commands can be used in the definition of a procedure:

```
User:      procdef copyit
Sys,User: 0000100 param alphname,dsorg,name1,name3
          0000200 ddef alphname,dsorg,name2
          0000300 ddef alphname,dsorg,name1
          0000400 cds name1,name2
          0000500 _correct 100
System:   PARAM ALPHNAME,DSORG,NAME1,NAME3
User:     *                                $2
Sys,User: excise 300
          insert 500
          0000500 default sysinx=e
          0000600 edit name2
          0000700 _end
```

While defining COPYIT, the user enters text on lines 100, 200, 300, and 400. When the system prompts with line 500, he decides to make a

correction in line 100. He enters a break 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 enters the correction. 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, 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.

The system then 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. The system again 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 next line to be entered as text, which, in this example, was line 500. (He could have entered INSERT LAST since he is adding to the end of the procedure.) 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 in a procedure definition follows:

```
User:       procdef pdef
Sys,User:   0000100 param alphname,vi,name1,name2,name3
            0000200 ddef dsname=alphname,vi,ddname=name1
            0000300 _ddef dsname=myprog,vs,ddname=test
                    _insert
            0000300 ddef dsname=name2,vi,name3
            0000400 _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 a break character followed by a DDEF command, which does 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 CLP (in this case, line 300). This use of the INSERT command is possible because the CLP was not changed by the issuance of a text-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 previously 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 prompts for the next command by issuing a break character.

For example, the user again wants to modify the procedure COPYIT, which he has previously defined. He enters:

```
procdef copyit
```

and the system prompts with the break character. The user may now enter a text-editing command or he may add to the procedure text by entering the INSERT LAST command.

To delete a procedure that has been defined, and to have the corresponding procedure name removed from the dictionary as a USERLIB entry, the user must enter the PROCDEF command, followed by the name of the procedure. He then enters a break character, followed by the EXCISE command, specifying the range of lines of the procedure as operands on the command. For example, to delete the previously defined procedure COPYIT:

```
User:      procdef copyit
Sys,User:  0000100 _excise 0,last
           end
```

EXCISE deletes the entire procedure from line 0 to the last line. BUILTIN, EDIT, END, PLI, or PROCDEF must be to complete the deletion of the procedure. Until one of these commands is issued, the reference to the procedure is not removed.

Note: Line 0 must be specified, since it is the line number assigned by the PROCDEF command to the procedure header.

DIAGNOSTIC MESSAGES DURING EXECUTION

Diagnostic messages that occur during the execution of a command procedure are output to SYSCUT, which may be a data set or terminal. When the diagnostic message requests the user to repair an error condition, the user can make the correction at his terminal, and the procedure continues executing. If the error is nonrecoverable, the diagnostic message is output to the SYSOUT data set. In nonconversational mode, the task is terminated, and diagnostic messages are sent to the SYSCUT data set. In conversational mode, the user is queried for a new command.

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 system-supplied commands or combination of them. The user creates an object program and defines it as a command by use of BUILTIN. (See the description of this command in Part III.)

As with a PROCDEF, the user 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 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 User Macro Instructions for a further description of these macros.

OPERAND RESOLUTION AND SUBSTITUTION

The user can specify operands for user-written commands that are 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. He specifies these parameters by entering, on the line following the PROCDEF command, the word PARAM followed by the dummy names he wants to have for the operands.

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/BPKDS 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/BPKDS macro instruction.

With user-written commands created with BUILTIN and PROCDEF, the parameters supplied as operands are resolved at the time the user-written commands are issued. Since system-supplied commands were created with either PROCDEF or BUILTIN, the description of the operand resolution and substitution process for user-defined commands pertains to system-supplied commands as well.

For commands created with PROCDEF, there is a procedure-expander routine that resolves operands and substitutes operands. Operand resolution consists of:

1. Analyses of calling operands
2. Analyses of procedure operands
3. Generation of operand equivalences

ANALYSIS OF CALLING AND PROCEDURE OPERANDS

When the user calls a procedure, he enters the procedure name and 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 and Keyword Notation

Following is a review of positional and keyword notation, which was introduced under "Operand Representation" in Part I.

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 BPKL 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 VISAM data set.

```
User:       procdef viddef
Sys,User:   0000100 param ddname,dsname
            0000200 ddef ddname,vi,dsname
            0000300 _end
```

This procedure call might be used:

```
User: viddef mybest,test1
```

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 because TEST1 is in the same position as DSNAME. The result of the above procedure call is:

```
DDEF MYBEST,VI,TEST1
```

For the same procedure, assume that the user wants to specify operands that are to be inserted into the DDEF command, but he wants to change the data set organization to VSAM. He enters:

```
User: viddef mybest,vs,test1
```

This procedure call is erroneous. By positional association of the calling operands and the PROCDEF PARAM line, these associations are made:

```
DDNAME=MYBEST
DSNAME=VS
```

The result of the above procedure call is:

```
DDEF MYBEST,VI,VS
```

MYBEST is the DDNAME; VI is still the data set organization, and VS is DSNAME. There could be a data set that is named VS, but the user intended TEST1 as the DSNAME.

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 and is shown in all-capital letters, followed by an equal sign, and value is the actual value of the operand.

Assume the following procedure defines a data set that dumps one or more data locations or expressions:

```
User:       procdef autodump
Sys,User:   0000100 param dsname=alphname,data='here:there'
            0000200 ddef ddname=pcsout,dsorg=vi,dsname=alphname
            0000300 dump here:there
            0000400 _end
```

The following procedure call can be used:

```
User:  autodump data='0:15r,0:6d,top:middle',dsname=myprg
```

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=MYPRG
DUMP 0:15R,0:6D,TOP:MIDDLE
```

This example also shows how a quoted string can be used to define several operands in one operand field. Although an operand in the DUMP command is not shown in keyword notation, it can be designated as keyword notation in the PARAM line and calling sequence. The operand 'HERE:-THERE' in line 100 of AUTCDUMP must be identical to the dummy operand specified in the DUMP command. The apostrophes are needed in the keyword expression only if special characters are used.

Defaults

The user can specify, alter, or delete default values for the operands of a user-written command in the same way he does with a system-supplied command. Unless the user has provided default values for the operands, these operands must be specified when the command is issued. The user creates a default value by issuing the DEFAULT command (see Section 6 and the DEFAULT command in Part III) and 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.

Example: This command procedure has been created to define a data set:

```
User:       procdef defcat
Sys,User:   0000100 param ddname=datname,dsname=alphname,dsorg=vp
            0000200 ddef ddname=datname,dsorg=vp,dsname=alphname
            0000300 _end
```

A dummy value of VP, which is not the default value of DSCRG (VI is the default), is assigned tc keyword DSORG.

Assume this sequence of commands in the procedure call:

```
User:       default dsorg=vs
Sys,User:   defcat testx,dsname=prog1,ddname=testxy
```

The result, after resolution of the values, is:

```
DDEF DDNAME=TESTXY,DSCRG=VS,DSNAME=PROG1
```

The following is an explanation of the way the system resolved the values that were entered:

1. DDNAME=TESTXY

   Although the dummy operand DDNAME=DATNAME is in keyword notation, it can also be considered positionally (that is, the first operand in the PARAM line). The system resolves TESTX as a possible value for DATNAME. However, DDNAME=TESTXY is specified in the calling sequence, so the system takes it as the value of DATNAME.

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 sets DSORG=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, and since the user has given a default value for DSORG, this value (VS) is assigned as a string to be substituted for VP in the PARAM line. Eventually, the system substitutes VS for the dummy operand VP in the DDEF command.

3. DSNAME=PROG1

   This operand is given in the procedure call in keyword format. PROG1 replaces ALPHNAME.

Nulls: The user can specify a null value for the operands of the user-written command. A null value is indicated by two successive apostrophes.

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 is 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 as:

   ```
   '', 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:

```
User:       procdef copy
Sys,User:   0000100 param dsname1,dsname2,base,incr
            0000200 if 'base'='''; cds dsname1,dsname2
            0000300 if 'base'¬='''; cds dsname1,dsname2,base,incr
            0000400 _end
```

Calling Sequence 1:

```
copy orig,dupe,base=''
```

Assume the user has specified a default value of 300 for BASE. The data set names ORIG and DUPE are 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 is substituted wherever the character string BASE appears in the text of COPY, as:

```
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 zero 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 is invoked.

Calling Sequence 2:

```
copy orig,dupe
```

Assume, again, that the user has 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 CRIG,DUPE
300 IF '300'¬='';CDS ORIG,DUPE,300,100
```

The conditions for the conditional statement in line 200 are not met (that is, '300' does not equal ''), but the conditions for the conditional statement in line 300 are met. The CDS command is executed.

GENERATION OF OPERAND EQUIVALENCES

The system establishes a table for the dummy operands, their corresponding keywords, and the calling sequence values.

The result that is generated is shown in Table 10. The first column, Internal String, contains the character string that is the dummy operand in the PARAM line. This dummy operand identifies the string that is to be replaced in the procedure text when substitution occurs. The second column, Keyword, contains the 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 is substituted later for the associated string in the Internal String column.

Table 10. Generation of operand equivalences

Internal String	Keyword	Value
String for which substitution occurs	Keyword in PARAM list	Value in calling sequence

Here is an example of how resolution of operands occurs, based on the process shown in Table 10. Assume this procedure has been defined:

```
User:       procdef asmwlist
Sys,User:   0000100 param alphname,stored=$n,lincr=(first,last),-
            0000200 version,symlist=$y
            0000300 asm alphname,stored=$n,lincr=(first,last),-
            0000400 verid=version,isd=y,symlist=$y,asmlist=y,-
            0000500 crlist=y,stedit=y,isdlist=y,pmdlist=y
            0000600 _end
```

This procedure call is made:

```
asmwlist myprog,stored=y,version=today,now,alphname=myprog1,symlist=n
```

The effect of operand resolution is shown in Table 11.

Table 11. 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 occurs	ALPHNAME	$N	(FIRST,LAST)	VERSION	$Y

For each string named, a value is 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 is taken. The system's table of operand equivalences looks 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, a substitution process occurs. The result of the procedure expansion, after operand substitution, is:

```
ASM MYPROG1,STORED=Y,LINCR=(100,100),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 is specified as $Y; the $ ensures that the calling sequence keyword value (N) associated with SYMLIST is 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 occurrence of string Y, string N is substituted. Since no LINCR operand was specified in the procedure call, the default value for LINCR is substituted.

Each character of the procedure text is compared with the first character of each internal string, beginning with the last internal string. If a matching character is found, the remaining characters of that particular internal string 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, comment, or delimiter within the text of the procedure.

When an entire internal string is matched, the characters of the procedure line are replaced by the calling value. When no match exists, the other first letters of internal strings 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 takes 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.

```
User:       procdef procall
Sys,User:   0000100 param op,funct,p='list'
            0000200 opfunct list
            0000300 _end
```

   Later, he invokes PROCLOAD:

```
prccall proc,load,'myprogx,1.0,h''50'''
```

   After operand resolution occurs, substitution takes place. OP, FUNCT, and LIST are replaced by PROC, LOAD, and 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. The following example shows what happens if dummy operand names are carelessly selected. Assume this procedure has been defined:

```
User:       procdef starter
Sys,User:   0000100 param progname,pari=a,b,c,r
            0000200 abacus name=progname,a,b,c,r
            0000300 _end
```

Later, a procedure call is issued.

```
starter zap,1.0,c=first,r=loca
```

After operand resolution and substitution, the result that is obtained is undesirable.

```
1.01.0FIRSTUS N1.0ME=ZAP,1.0,,FIRST,LOCA
```

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 as follows:

```
User:       procdef starter
Sys,User:   0000100 param progname,pari=$a,$b,$c,$r
            0000200 abacus name=progname,$a,$b,$c,$r
            0000300 _end
```

The procedure call is issued as before:

```
starter zap,1.0,$c=first,$r=loca
```

The result is:

```
APACUS ZAP,1.0,,FIRST,LOCA
```

3. This example illustrates that substitution occurs on full matches with operands in a right-to-left occurrence. Assume that two procedures have been defined.

```
User:       procdef fake1
Sys,User:   0000100 param ab,abc,abce,c,e
            0000200 if 'ab'='abc'; display 'abce'
            0000300 _end
            procdef fake2
            0000100 param abce,abc,ab,c,e
            0000200 if 'ab'='abc'; display 'abce'
            0000300 _end
```

The calls made are:

```
fake1 loca,no,cond,c='',e=code

fake2 loca,no,cond,c='',e=code
```

After operand resolution and substitution:

```
FAKE1
IF 'LOCA'='NO'; DISPLAY 'COND'

FAKE2
IF 'COND'='COND'; DISPLAY 'CONDCODE'
```

In the call to FAKE1 there is no output. In the call to FAKE2 the system prints out CONCODE.

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.

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 examples below illustrate PROCDEF usage.

1. The user wants to combine the DISPLAY and DUMP commands, and he wants 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:

```
User:       procdef output
Sys,User:   0000100 param alter,data1,data2,data3,dsname
            0000200 if 'alter'=''|'alter'='y';display data1,-
            0000300 data2,data3
            0000400 if 'alter'='n';ddef pcsout,vi,dsname;dump-
            0000500 data1,data2,data3
            0000600 _end
```

   ALTER serves as a switch; if 'Y' is specified, or if ALTER is omitted, DISPLAY is executed. If 'N' is specified, DDEF and DUMP are executed. The user may execute a DUMP to display two data fields at the printer with the following calling procedure:

```
output alter=n,data1=field1,data2=field2,dsname=data
```

   Only line 300 of OUTPUT is executed, which 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 200 of OUTPUT 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 the UPDATE command. He defines a procedure:

```
User:       procdef change
Sys,User:   0000100 param $ds,$rn
            0000200 default sysinx=e; edit $ds,rname=$rn
            0000300 __default sysinx=g; update
            0000400 _end
```

   If the user wants to update a region in a data set, and the text editor is not invoked, he invokes his procedure CHANGE. For example, to update the region XYZ in the data set MYDATA:

```
change mydata,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 to be an option of either SYNONYM or DEFAULT:

```
User:       procdef def
Sys,User:   0000100 param par,spec,csw,save
            0000200 default par=spec
            0000300 if 'save'='y'; profile csw
```

```
0000400 _end
procdef syn
0000100 param term,string,csw,save
0000200 synonym term=string
0000300 if 'save'='y'; profile csw
0000400 _end
```

The user can 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 add a message to his message file:

```
User:       procdef bldmsg
Sys,User:   0000100 param msgid=$1,text=$2
            0000200 default sysinx=e
            0000300 edit userlib(sysmlf),rname=$1
            0000400 __update
            0000500 0 $2
            0000600 __end
            0000700 display 'msg $1 filed'
            0000800 _end
```

Since the EDIT command is included in this procedure, with a member name of SYSMLF following the DSNAME, the user always gains access to his message file by issuing BLDMSG.

5. The user wants to have the PARAM line automatically built after issuing the PROCDEF command:

```
User:       procdef noparam
Sys,User:   0000100 param pname,$1,$2,$3,$4
            0000200 default sysinx=e;procdef pname
            0000300 param $1,$2,$3,$4
            0000400 __default sysinx=g;insert
            0000500 _end
```

The user can issue a PROCDEF and PARAM on one line

```
noparam pname,dpar1,dpar2,dpar3,dpar4
```

and will be prompted to enter the lines for the text of the procedure.

6. The user wants to generate a command to eliminate previously defined procedures:

```
User:       procdef destroy
Sys,User:   0000100 param @x
            0000200 default sysinx=e; procdef @x
            0000300 __excise 0,last;end;display '@x eliminated'
            0000400 _end
```

To eliminate OUTPUT, which was defined by PROCDEF in example 1, the user issues

```
destroy 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 can define the actions that he wants performed by ZLOGON with either

PROCDEF, BUILTIN, or SYNCNYM. For example, a user who uses only one program might create this procedure:

```
User:       procdef zlogon
Sys,User:   0000100 qualify pyroll
            0000200 pyroll
            0000300 _end
```

Every time he initiates a task, the system qualifies all internal symbols implicitly, loads his program, and causes the program to start execution. (PYROLL must reside in his USERLIB in order to be loaded and executed at this time.)

8. The user defines a procedure that ascertains current default values for operands in his user profile:

```
User:       procdef def?
Sys,User:   0000100 param $x
            0000200 default sysinx=e
            0000300 procdef demo
            0000400 param $x=$y
            0000500 display '$x=$y'
            0000600 __demo
            0000700 excise 0,last
            0000800 end
            0000900 default sysinx=g
            0001000 _end
```

Now, if the user enters

```
def? lineno
```

the system responds with

```
LINENO=Y
```

During a user's task he is likely to receive any number of system messages that inform him of errors, request necessary information, or describe the status of some requested operation. These messages are normally issued from the system message file (the SYSMLF member of SYSLIB). When the proper conditions arise, the system calls the user prompter to display a message from the message file.

MESSAGE GENERATION AND RECEPTION

The user can control the messages that he receives: he can reset the LIMEN and BREVITY operands to screen out unwanted messages or to specify the length of messages he receives; he can change the text of system messages; and he can add new messages to be issued from his own PROCDEFs and BUILTIN-defined commands. These operations, described later in this section, are summarized below.

1. To screen out unwanted messages or message IDs, the user can reset the LIMEN and BREVITY implicit operands by using the DEFAULT command (as described in Section 6). Initially, the user receives standard messages without the message ID codes (BREVITY=T). These messages are issued when an error has occurred, and the user must know about the error (LIMEN=W).

2. The user can change the text of system messages. He must create a message file in his own USERLIB (again, member SYSMLF). Then he can put the new form of the system message in his own message file. Since the system searches the USERLIB message file before the system message file, the user's altered form is displayed. This procedure is described later in this section.

3. The user can put new messages in his message file and have the user prompter display them from PROCDEFs or BUILTIN-defined commands. This procedure uses the PRMPT command or PRMPT macro instruction and is described later in this section.

When the user creates a message file containing his own versions of messages issued from the system message file, or his own messages for user-written commands, other users are not affected because any changes made to one user's USERLIB do not change another user's task.

Message Explanation

The user can issue an EXPLAIN command that causes the system to give an explanation of a system message or of specific words within a system message. (See Part III for a description of this command.)

Message Generation

The user can generate a message by calling the user prompter. He issues either the PRMPT command (see the command description in Part III) or the PRMPT macro instruction (for assembler language programs; see Assembler User Macro Instructions) to invoke the user prompter.

Message Filtering

Message filtering is the process of determining which messages the system displays. Each message is classified in each of three categories when it is created. The three categories, as shown in Table 12, are:

1. Severity of the message (LIMEN; see Part A of the table)

2. Length and type of message (BREVITY; see Part B of the table)

3. Mode of the user's task (see Part C of the table)

SEVERITY: LIMEN is the operand name in the user profile for the severity of the message. The severity codes shown in Table 14 (Part A) are in order of increasing severity. The user prompter does not display messages that have a severity code lower than the value of LIMEN. For example, if the filter code is X, all X and T messages are displayed; however, I and W messages are not. The user can change the system default for LIMEN (see Appendix C for system defaults) by issuing a DEFAULT command with the new LIMEN value. (See "Implicit Operands" in Section 6 for the possible values of LIMEN and Part III for a description of the DEFAULT command.)

LENGTH AND TYPE: BREVITY is the operand name for message length and type. There are six different classifications for this operand. (See Table 12, Part B.) The shortest message is the message ID only. The standard message is the message ID and the message text. (The system supplies this version in the system message file.) The extended message is an alternate message that is created at the installation and is used instead of the standard message. The user can specify that standard or extended messages be issued without their message ID. A reference message points to another message that contains the message text. The user prompter displays a message according to the value of BREVITY that is specified in the user profile. To alter the system default for message length and type, the user issues a DEFAULT command with the desired value for BREVITY. (Note: R is not a default value for BREVITY, but is specified when the user creates a message that he wants to designate as a reference message. See the discussion on reference messages later in this section.)

MODE: When a user is creating a message, he can specify that it be issued for conversational tasks, nonconversational tasks, or both. He cannot modify this specification with the DEFAULT command. He specifies C, B, or A where: C messages are displayed only for conversational tasks, B messages only for nonconversational tasks, and A messages for all tasks, regardless of mode.

Table 12. Filter codes

A. LIMEN	Code	B. BREVITY	Code	C. Mode of Task	Code
Information	I	Message ID	M	Conversational	C
Warning	W	Standard	S	Nonconversational	B
Serious Error	X	Extended	E	All	A
Terminate Error	T	Standard, no ID	T		
		Extended, no ID	X		
		Reference	R		

MESSAGE FILE CONSTRUCTION

The user's message file is a member of USERLIB; this member is called SYSMLF. Using the text-editing commands, the user can construct and maintain his own message file. To initiate this process, he enters:

```
edit userlib(sysmlf),z0047
```

The second parameter, Z0047, is the message ID for the new message that is being added to the message file. (The message ID can be from one to eight characters.) Notice that the parameter occupies the RNAME (for region name) position in the EDIT command. Each message is contained in its own region in SYSMLF; the region name is the same as the message ID for the message. Now, since this is a new region in the SYSMLF data set, the system prompts with the first line number, line 100:

```
User:    edit userlib(sysmlf),z0047
System:  0000100
```

Since standard messages are stored at line 0 (see "Message Type and Format" later in this section), the user wants to put his new message at line 0:

```
Sys,User:  0000100 _update
```

(No prompt is issued by UPDATE. The keyboard is unlocked when the user can enter data. The user enters the following lines.)

```
User:  0 wsa this is the message text
       _end
```

The underscore at line 100 tells the editor that the user wants to enter a command; he enters the UPDATE command. Then, he enters the 0 (for line 0), one blank only, the message filter code (WSA), one blank (more than one blank is included as part of the message text), and the message text. The user has created message Z0047; the message reads, "this is the message text"; the filter code (WSA) means: W--a warning message, S--a standard message, and A--a message for all of his tasks, whether conversational or nonconversational.

The user may require more than one line for the text of his message. He has issued the EDIT command for the existing message ID, Z0047 (see above), and the system has responded with line 100. He then enters:

```
Sys,User:  0000100 _update
           0 wsa this message requires more than one line -
           10 wsa this is the continuation of the message
           _end
```

In this example, the user has typed a continuation character (hyphen) at the end of line zero. (A blank precedes the hyphen so that the text of the lines will not be run together.) When the keyboard is unlocked, the user enters the next line, which can start with any line number from 1 to 99. The next line must be in the same format as the previous line.

Many messages require that variable text (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 $NN, where N can assume the integer values 1 through 20 and denotes the Nth element of the parameter sublist in the calling sequence. (Refer to the PRMPT macro in Assembler User's Macro Instructions.)

REFERENCE MESSAGE

The user prompter can be used to reduce the number of times a message may appear in the message file and still maintain unique message IDs for every distinct call to a message. For example, if user message XYZ is established with text identical to the text of system message ABC, message XYZ can point to ABC rather than repeat the text of ABC. Message XYZ is created as a reference message.

To use a reference message the user must:

- Put an R in the second position of the message classification code (for example, WRA) for a message that is referencing another message. One blank must follow the message classification code.

- Put the ID of the referenced message in the eight positions that immediately follow the blank. Blanks must be added to the end of the message ID if the ID does not occupy all eight positions.

- Put the line number of the references message in the seven positions of the line that immediately follow the message ID.

Example: The construction of a reference message is as follows:

```
WRAbABCbbbbb0000000
```

(In this example, the symbol b indicates a blank.) This causes a reference to line 0 of message ABC whenever message XYZ is issued. A user is limited to 15 reference pointers in one chain in locating a desired text.

MESSAGE TYPES AND FORMAT

There are five types of messages in the message file: standard, extended, response, explanation, and word explanation. All or any of these types can be used for each message ID. (Note: in the examples below, one blank must separate the message classification code from the line number, and one blank separates the line number from the message text.)

- Standard message is a brief communication from a program to the user. A standard message always begins on line 0, and continuation lines can be on lines 1-99. (See the examples under "Message File Construction" above.)

- Extended message is an alternate message that is issued instead of the standard message. An extended message begins on line 100 and can be continued on lines 101-199. For example, the user wants to add an extended message for the existing message ID Z0047. He types in:

```
User:       edit userlib(sysmlf),z0047
Sys,User:   update
            100 wea this is an alternate message -
            105 wea to be used in place of the -
            110 wea standard message when brevity -
            115 wea has been defaulted to e.
            _end
```

If the user wishes to reference the text of the extended message for XYZ005, instead of Z0047, he enters:

```
100 wra xyz005bb0000100
```

(You should make special note of the R in the second position of the classification code and of the reference to line 0000100, where the text of the extended message is located.)

- Response message can be in one of the three forms listed below. The response line begins on line 200 and can be continued on lines 201-299. Forms of response message:

1. Text that states the possible responses to the message. This line is used for messages with unpredictable responses, and it is issued when an EXPLAIN RESPONSE is entered by the user.

(SeePart III.) To add a response line for existing message ID MN041, which expects a response, the user enters:

```
User:       edit userlib(sysmlf),mn041
Sys,User:   update
            200 wsa this line explains the-
            205 wsa expected response for -
            210 wsa message mn041
            _end
```

When the user issues EXPLAIN RESPONSE, the system displays the text of line 200 and any continuation lines (in this case, lines 205 and 210).

2. A set of predictable response words (for example, yes or no) and their associated response codes. Each predictable response word must have, as a response code, a unique positive integer. A response must be in the form

   word1=code1,word2=code2,...,wordn=coden

   and can be entered as follows (assume that the user has issued UPDATE and the keyboard is unlocked):

   200 wsa yes=1,no=2

   The codes are used by the user prompter as return codes to the issuing program to indicate which response was entered by the user. For the example just given, if the user issues EXPLAIN RESPONSE after a message is displayed, the system prints out:

   VALID REPONSES ARE: YES,NO

3. A response line can reference the response line (line 200) of another message. It must be set up in the same format as all reference messages (see the discussion on reference messages, above).

- Explanation message makes clear a standard message or an extended message. An explanation message begins on line 300 and can be continued on lines 301-399. For example, the user wishes to add an explanation message for the existing message ID Z0047. He enters:

```
User:   edit userlib(sysmlf),Z0047
        update
        300 wsa this line is displayed -
        305 wsa when the user enters -
        310 wsa explain immediately after the -
        315 wsa standard message at his terminal
        _end
```

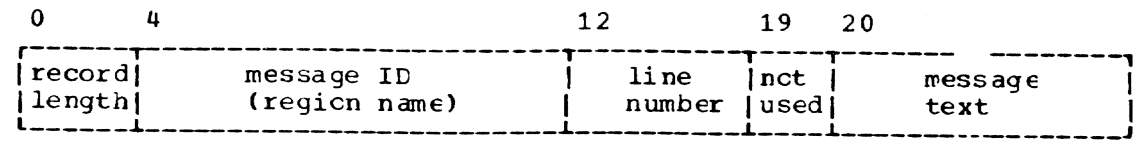
- Word explanation clarifies one or more words in a message. Word explanation messages begin on line 400 and above. For example, the user enters:

```
User:   edit userlib(sysmlf),fff001
        0000100 _update
        0 wsa this message contains references to word1 and word2
        400 wsa word1bbbthese lines are for explanaticns
        500 wsa word2bbbof words that appear in the standard message
        _end
```

The first eight positions of the text are for the word to be explained, and the rest of the line is for the explanation. When the EXPLAIN command is issued, with a specific word as the operand,

immediately after a standard message is displayed, the lines starting with line 400 are searched to see if an explanation is available for the requested word.

The format of a VISAM variable-length record is:



The format of the message text (bytes 20-255) varies with the type of message. For all types of messages bytes 20-22 contain a message classification code and byte 23 is blank. Bytes 24 to the end of the line contain different information, as shown in Table 13. You can use one of the numbered items (for each message type) shown under the content of the message.

Table 13. Message content

Message Type	Content (Byte 24 to End of the Line)
Standard (Line 0)	(1) Message text (2) Bytes 24-31 Reference message ID Bytes 32-38 Reference line number Bytes 39 on Unused
Extended (Line 100)	(1) Message text (2) Bytes 24-31 Reference message ID Bytes 32-38 Reference line number Bytes 39 on Unused
Response (Line 200)	(1) Explanation of response to be entered (2) word1=code1,word2=code2,...,wordn=coden (3) Bytes 24-31 Reference message ID Bytes 32-38 Reference line number Bytes 39 on Unused
Message Explanation (Line 300)	(1) Message explanation text (2) Bytes 24-31 Reference message ID Bytes 32-38 Reference line number Bytes 39 on Unused
Word Explanation (Line 400)	(1) Bytes 24-31 Word to be explained Bytes 32 on Word explanation text (2) Bytes 24-31 Word to be explained Bytes 32-39 Reference message ID Bytes 40-46 Reference line number Byte 47 Blank Bytes 48-55 Reference word Bytes 56 on Unused
Note: The user must observe strict byte alignment in message fields.	

Word Explanation Scope

The scope of word explanations varies, depending on the word. For example, some word explanations are universal in scope. Other word explanations are universal within a major component of TSS, such as the command system. Still others are limited to the particular message in which they appear. Broad scopes are an advantage because fewer explanation records are needed, which reduces the size of the message file. The user can regulate the scope of word explanations to favor his particular operation.

The scope of a word explanation is indicated by an eight-byte message ID. Explanations with universal scope have an all-blank message ID (that is, a blank region name). For a scope restricted to a single message, the full eight-byte message ID is used. Identification codes can be assigned in a pattern to allow various levels of scope, between universal and fully restrictive. All message IDs for the command system start with CZA. 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, which are unique within the module.

The user prompter has two one-byte masks that allow users to control the user prompter's search for word explanations. One mask is for the system message file, and one is for the user message file. Each bit in the mask corresponds to one position in the ID of the message that contains the explainable word. For example, bits 0-7 in the scope mask correspond to positions 1-8 in the message ID. The scope mask indicates how many positions in the message ID are to be compared when a search is made for a word-explantion record. The user prompter scans the mask, starting on the right with bit 7. It looks for bits set to 1. Each 1 bit found causes an access to the message file. The message file is searched for the message ID, beginning with position 1 and continuing through the position that corresponds to the mask bit that is one.

For example, bits 7, 4, and 2 are turned on in the user scope mask. A message with the ID ABCXX200 is issued. This message has an explainable word, DSNAME, and the user issues EXPLAIN DSNAME. In the search, all eight positions of the message ID are used first because bit 7 is on; therefore, lines 400 and above in the region ABCXX200 are searched for the word DSNAME. Next, the lines 400 and above in the region ABCXX, if it exists, are searched because bit 4 is on; and then the lines 400 and above in the region ABC, if it exists, are searched because bit 2 is on. The all-blank region is searched last. If DSNAME is not found in the user message file or in the system message file, a diagnostic message is issued that indicates that no explanation is available.

Both the system mask and the user mask are located in the character switch table, which is a section of the user profile (see Appendix C). The user scope mask may be changed by using the MCAST command. The system mask and the user mask are set initially, as a part of the prototype profile.

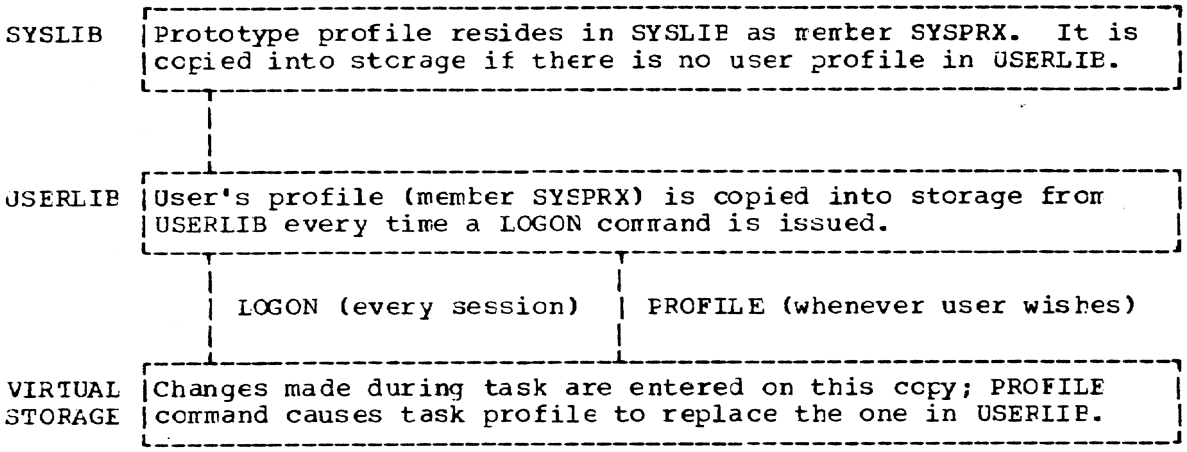
SECTION 6: THE USER PROFILE

The user profile is the data set that controls the user's operating environment. It contains the default values for omitted operands, the values for implicit operands, synonyms that the user creates for command names or operands, command symbols established with the PCS SET command (see Section 3), and a pair of character translation tables -- one for input, and one for output.

When a user is first connected to TSS, he is provided with a prototype profile; this contains only system-supplied default values, implicit operand values, and the translation tables (including certain miscellaneous control characters that are described in Appendix C). Then, when the user first initiates a task (via LOGON), the system searches his user library (USERLIB) for a user profile (member SYSPRX). If SYSPRX cannot be found -- this occurs when the user first logs on to the system or after he erases his profile -- the system uses the prototype profile member (SYSPRX) as the task profile.

The user creates a user profile by issuing a PROFILE command. Then, when he logs on later, the SYSPRX member in his USERLIB is used to create his task profile. The prototype profile is not used as long as there is a USERLIB (SYSPRX) member for this user. The user can erase his user profile by issuing the following command:

```
ERASE USERLIB(SYSPRX)
```



The task profile controls the user's operating environment. The user can alter his task profile with the SYNONYM, DEFAULT, MCAST, and MCASTAB commands; then he can make the changes a part of his permanent user profile with the PROFILE command. The user profile management commands are summarized in Table 14.

Table 14. User profile management commands

Command	Function
SYNONYM	Rename commands, keywords, PCS operands, or command statements.
DEFAULT	Add, replace, or delete entries in default table.
PROFILE	Make changes to task profile permanent in user profile.
MCAST	Alter miscellaneous control characters.
MCASTAB	Alter input and output translation tables.

SYNONYMS AND DEFAULTS

The user can alter default values with the DEFAULT command. Default values are used by the system when operands needed by the system are not included when the user enters a command. Appendix C shows the default values for the system operands. Note that not all operands have system-supplied default values.

To change the default value for an operand, the user enters the DEFAULT command giving the name of the operand and the new value:

```
default regsize=7
```

The default value for the REGSIZE operand is now set to 7 for the remainder of the task, or until the user again alters it. This new value is set in the task profile.

The user can rename keyword operands or commands with the SYNONYM command. He enters the new name, an equal sign, and the old name:

```
synonym p=pc?
```

Now, when the user enters the P command, the system executes the PC? command.

To remove the effects of a SYNONYM or DEFAULT command, the user enters the command this way:

```
synonym p=(press RETURN)
default regsize=(press RETURN)
```

Since no value appears to the right of the equal sign in these examples, the current settings are destroyed. Changes made with SYNONYM and DEFAULT in this way are made to the task profile. If the user logs off and again logs on, the values in the new task profile are taken from the user profile or the prototype profile, neither of which was changed.

PROFILE COMMAND

To make changes permanent, the user issues the PROFILE command after the DEFAULT or SYNONYM command. For example:

```
default regsize=7
profile
```

Now, the value of REGSIZE is 7 in subsequent tasks until the user alters it. If the value is changed later, the user can make that value permanent with the PROFILE command. For example:

```
default regsize=(press RETURN)
profile
```

IMPLICIT OPERANDS

Implicit operands are not entered with any command. Rather, they control certain aspects of the user's operating environment. For example, the LINENO implicit operand indicates whether the user wants the text editor to prompt with line numbers or to unlock the keyboard and do no prompting. The system value is Y (line numbers are displayed). If the user does not want these line numbers (especially when a data set is be-

ing listed at the terminal) he can change the value of LINENO with the DEFAULT command, just as he changed the value of a command operand.

```
default lineno=n
```

Again, this change is made only in the task profile; the user can then issue the PROFILE command to make the change permanent in the user profile.

Default values for command operands and implicit operands are shown in Appendix C. Table 15 lists the implicit operands and their functions.

Table 15. Implicit operands

Operand	Function	System Default Value	Other Values
ALPHABET	controls character set used at the terminal	1 - folded mode	2 - full EBCDIC 3 - PTTC/6 4 - PTTC/8
BREVITY	message length	T - standard without message ID	M - message ID E - extended message S - standard message X - extended without message ID
CLEANUP	controls cleanup of user attention-handling when EXIT is issued for a level 1 program	Y - user attention-handling is cleaned up	N - user attention-handling is ignored
CONPRMPT	controls record-concatenation prompt character	Y - prompt character is issued	N - no prompt character
CONREC	controls record-concatenation processing	N - no record-concatenation processing	Y - record-concatenation processing is done
DEPROMPT	prompting during ERASE and DELETE	Y - prompt for disposition	N - no prompt
DIAGREG	controls display of registers during ABEND	N - registers are not displayed	Y - registers are displayed
HEXSW	control characters used to indicate hexadecimal processing	x% - indicates hexadecimal input follows	user defined
LIMEN	message severity	W - warning message	I - information X - serious error T - terminate error
LINENO	controls line number issuance	Y - line numbers are issued	N - no line numbers are issued
ASMALIGN	controls alignment of source code in assembler list data set	Y - source code aligned	N - source code not aligned

PROGRAM PRODUCTS UNDER TSS

TSS allows the execution of OS/VS compilers via interface modules. The compilers themselves are not changed; restructuring of the object code after the program product is installed on a TSS system is done to allow the program product to execute under TSS. To provide suitable interfaces, a certain degree of OS/VS simulation has been implemented.

The sequential, index-sequential, direct, and partitioned access methods are logically simulated; the data records are physically maintained in TSS formatted data sets and are processed internally to simulate OS/VS data set characteristics.

OS/VS Supervisor Call functions such as GETMAIN/FREEMAIN and TIME are simulated at the functional level.

The simulation restrictions on OS/VS object programs executing under TSS are primarily related to VSAM, and telecommunications access methods. Functions related to multitasking are ignored. TSS restrictions remain in effect.

PROGRAM PRODUCTS SUPPORTED

The IBM Program products supported are as follows:

5734-AS1 Assembler H

5740-CP1 VS COBOL Compiler & Library

5734-FO3 FORTRAN IV (H Extended) Compiler

5734-LM3 FORTRAN IV Library Mod II

5734-PL1 PL/I Optimizing Compiler

5734-LM4 P L/I Resident Library

5734-LM5 PL/I Transient Library

5734-PL3 PL/I Optimizing Compiler and Libraries.

Note: Installation of these program products occurs using the PPREAD command (described in the System Programmer's Guide) and installation scripts supplied.

PROGRAM PRODUCT LANGUAGE INTERFACE COMMANDS

The PPLI commands are as follows:

COBOL FILEREL HASM CSDD? PLICPT

FILEDEF FTNH ODC OSRUN

These commands are discussed in alphabetical sequence in the Command Section of this manual.

PART III: COMMAND DESCRIPTIONS

This part contains format illustrations, descriptions, and examples of the use of the commands. The commands appear in alphabetical order. The symbol is used in the left-hand margin to help you find a command.

If you do not need detailed information on the format of a command, but need only a review of the operands, you should turn to Appendix G. You should review Part I before you look at the command descriptions if you are not familiar with the way commands are described in this book.

The format illustration of some commands show all operands within brackets ([ ]). This indicates that you do not have to enter any operand with the command. The action that the system takes when you do not specify an operand is discussed with the command. If an operand can be entered in keyword format, the keyword is shown in all-capital letters. If an operand is in lowercase letters, you cannot use keyword notation.

ABEND Command

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

Operation	Operand
ABEND	

Note: There are no operands.

Functional Description: When ABEND is executed, the current task is terminated. A new task is created, as if you had issued another LOGON command. All data set definitions, open data sets, and task variables are eliminated.

Note: If you issue the command DEFAULT DIAGREG=Y before you issue ABEND, the system displays general register contents following the ABEND. The system default for DIAGREG is N.

Example: Termination, using ABEND, is as follows:

```
User:     (press ATTENTION key once)
System:   !
User:     abend
System:   TASK DELETED BY COMMAND
          NEW TASK LOGGED ON AT 11:12 ON 06/09/71.  TASKID = 002D
```

ABENDREG Command

This command displays the contents of general registers when ABEND occurred and the location within your task where the ABEND occurred.

Operation	Operand
ABENDREG	

Note: There are no operands.

Functional Description: After your task has been abnormally terminated by the system, or after you have entered the ABEND command, you may use the ABENDREG command to display the general registers at the time termination occurred. No display occurs if your task has not been terminated abnormally.

Example: Your task has just been abnormally terminated, and you want to see the contents of the general registers at termination:

```
User:    abendreg
System:
        ABEND IN PRIV PRG CZASBC  +000902, LAST USER LOC AT CFALBC  +0003E8
        USER GRS 003C47A8 00007438 00113000 0001177A 00006BC0 00008058 00112000 00000017
                 00008054 00539DEC 0000804C 00006200 00011000 00007020 001125D8 00000008
        PRIV GRS 000000FF 000598F8 00000008 00000048 00011740 00135E76 001354B8 0001C549
                 00000000 00001000 0042F0BC 001C68AC 000593A8 00059440 001C6902 001BC000
```

ASM Command

This command invokes the assembler to assemble a source program module.

Operation	Operand
ASM	NAME=module name[,STORED={Y|N}] [,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|E}] [,STEDIT={Y|N}][,ISDLIST={Y|N}][,PMDLIST={Y|N}] [,LISTDS={Y|N}][,LINCR=(first line number,increment)]

NAME
identifies the object module to be created.

If the source program module (that is, the source language data set) is prestored, the user must have named it SOURCE.name. If it is not prestored, the system automatically prefixes SOURCE. to the source program module name. The listing data set is automatically 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 name from one to eight alphameric characters long. The first character must be alphabetic. The object module name must be unique to the library in which it is stored. See Assembler Programmer's Guide for a complete list of naming rules.

STORED
specifies whether or not the source program module is prestored (that is, whether or not the data set SOURCE.name exists).

Specified as:

Y - source program is prestored.
N - source program is not prestored.

System default: N.

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 DDEF commands within the current task. The user can specify a maximum of six libraries (that is, six pairs of data

definition names), which are searched in the opposite order in which they were specified; the system macro library (SYSMAC) is made available to the user automatically and is searched last.

Specified as: the data definition names defined in the 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: from one to eight alphameric characters.

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.

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.

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.

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.
E - cross-reference listing of only the symbols actually used is produced.

System default: N.

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.

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.

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.

LISTDS
determines whether the user-requested listings from the assembler are to be placed in a list data set or are to be placed directly on SYSOUT.

Specified as:

Y - listings are placed in list data set.
N - listings to SYSOUT.

System default: Y.

LINCR
specifies the first line number of the source language data set and the increment to be applied to get succeeding line numbers.

Specified as: two three- to seven-digit decimal numbers, separated by a comma and enclosed in parentheses; the last two digits in each number must be zeros.

System default: (100,100).

Note: This operand is ignored when STORED=Y.

There is also an operand called the ASMALIGN operand. ASMALIGN is not an operand of the ASM command, but is instead an implicit operand (Section 6, Part II explains implicit operands). ASMALIGN controls the alignment of the source statements in the assembler list data set.

If you issue DEFAULT ASMALIGN=Y prior to issuing the ASM command, all names, operation codes, and operands in your source code will be aligned in columns 1, 10, and 16 (respectively) in the list data set. ASMALIGN=Y is the system default; if you desire alignment and haven't specified otherwise, alignment is automatic. However if you do not want your source statements aligned, issue DEFAULT ASMALIGN=N prior to issuing the ASM command, and your source statements will appear in the list data set just as you entered them.

Functional Description: See "Language Processing" in Section 3 of Part II.

Caution: The command is canceled if invalid operands are entered.

Examples: (refer also to Assembler Programmer's Guide)

1. The user wants to assemble a prestored source program (SOURCE.IRISH); he wants an ISD and a source program listing:

   User: asm irish,y,isd=y

   The system assembles the program SOURCE.IRISH and acknowledges successful assembly by prompting with an underscore.

2. The user wants to assemble a program as he enters it. The commands and data he enters from the terminal are as follows:

   Sys,User: asm tester,n,symlist=y

```
0000100         save      (14,12)
0000200         l         14,71(0,13)
0000300         st        14,8(0,13)
0000400         st        13,4(0,14
0000400 E *** OPERAND  FIELD IMPROPERLY DELIMITED
0000400         ST        13,4(0,14
#400,           st        13,4(0,14)
#
0000500         lr        13,%
0000500 E *** STATEMENT CONTAINS INVALID CHARACTER
0000500         IR        13,%
#500,           lr        13,14
#
.
.
.
0002600         end
MODIFICATIONS?
n
0000200  W *** OPERAND REQUIRES FULL-WORD BOUNDARY
MINOR ERRORS
```

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
         edit mylib

   The user creates a VISAM data set. He will use the symbol) as a header flag character.

   Sys,User: 0000100)macro1

   Line 100 is the header of MACRO1. Lines 200 through 600, which are the text of MACRO1, are not shown.

   0000700)macro2

   Line 700 is the header of MACRO2. Lines 800 through 1500, which are text of MACRO2, are not shown. The user ends processing of the EDIT command (see EDIT) as follows:

```
00001600 _end
```

Next, the user defines a macro instruction library index and calls SYSINDEX.

```
User:   ddef index,vs,myndx
        sysindex
```

The system prompts the user for control statements.

```
Sys,User:  header=),length=8
           asm myprog,y, (source, index)
```

The user assembles his program.

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[,...]

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. A command statement containing an AT is called a dynamic statement. Only one AT may be included in a dynamic statement, and it must be the first command in the statement. (See "Use of Command Statements" in Section 3 of Part II. Note the list of commands that can be used after AT.) The system assigns a number to each dynamic statement. This number may be referenced by the REMCVE command.

When an AT command is executed, a standard output (including the instruction location where the command became effective, program status information, and the statement number is presented to the user. If LIMEN is not set to I, only the dynamic statement number is displayed. If the AT is a conditional statement, the dynamic statement number is displayed only if the condition is true. 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 in 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 (SVC) 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 one when the program arrives at an instruction location designated in the AT command. The counter is incremented even when the dynamic statement is conditional if the specified location is reached. The counter may be used as an operand in the other PCS commands within the statement. The AT command alone will

interrupt, but not stop, program execution. (See "Types of Operand Specifications" in Section 3 of Part II.)

Caution: The user should not designate an instruction location that was modified by program execution. If he does, the results are unpredictable. Also, since PCS only checks that the AT location is on a halfword boundary, the user must be careful to put the AT command at the beginning of an instruction, not in the middle.

Programming Notes: If AT specifies FORTRAN statement numbers as instruction locations, the numbers must only designate executable FORTRAN statements.

Example: The user wants to be informed when his program reaches the locations PGM.S1, PGM.S3.(4), FTNPGM.98, and FTNPGM.98(5).

To accomplish this,

```
User:     at pgm.s1,pgm.s3.(4),ftnpgm.98,ftnpgm.98(5)
System:   00001
```

Execution of the program begins. When control arrives at any of the instruction locations, the user is notified. For example, the system prints out the following line (assuming LIMEN=I) when it reaches the third location specified in the command:

```
System:   AT FTNPGM.98 PSW  1  3  0  0003F076 0001
```

In this statement

FTNPGM.98 is the instruction location
PSW 1 3 0 003F076 is the program status
0001 is the statement number assigned by the system

Note: If LIMEN had not been set to I, only 0001 would have been printed by the system.

BACK Command

This command converts the user's conversational task to a nonconversational task.

Operation	Operand
BACK	DSNAME=data set name

DSNAME
identifies the cataloged VSAM or VISAM line data set (new SYSIN) that contains the series of commands that complete the current task in nonconversational mode.

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 to the task. Control of the task is passed to a new SYSIN. The nonconversational task takes its commands from the SYSIN data set named in the BACK operand field. The SYSIN data set should conclude with a LOGOFF command; if it does 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. This allows the user to continue

his task in conversational mode, as though he had not issued the BACK command.

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

Caution: If private devices are needed by a nonconversational task that is initiated by the BACK command, each device must be either: (1) assigned to the task by an active data definition (DDEF), or (2) reserved by a SECURE command, which must be the first command (other than GC) of the SYSIN dataset.

Programming Notes: The BACK command is ignored by the system when it is issued by a nonconversational task.

After issuing the BACK command, the user must re-issue the LOGON command to begin a new conversational task.

If the BACK command is rejected, the user can re-issue the command later. It may be necessary to first modify the new SYSIN data set to reflect any further conversational processing that has been done.

If the user interrupts a program that is being executed when he issues the BACK command, the first command in his SYSIN data set should be GC, which causes 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 the data set ALPHA as SYSIN for his nonconversational task. He issues the BACK command as follows:

```
User:    back alpha
System:  BSN=0001
         TERMINAL LOGICALLY DISCONNECTED, RECONNECT OR HANG UP
```

BEGIN Command

This command connects the user's task to an MTT application program running under TSS.

Operation	Operands
BEGIN	application name[,application parameters]

application name
: specifies the user-written application program name.

  Specified as: from one to eight alphameric characters.

application parameters
: specifies the user-written operand parameters (if any) that are entered according to the requirements of the application.

Note: The application program must define a means by which its current users may elect to be disconnected. Once the user is connected to the application program, any commands that have been defined by the application program can be entered.

BLIP Command

The 'BLIP' command allows the user to receive assurance that the system is still active and the terminal is connected. This command is only valid for 2741's or their equivalent.

Operation	Operand
BLIP	TIME=,*READ=

TIME
the decimal value given is the number of seconds between the signals to the user. If 0 is entered, no assurance signal will be given by the system.

Specified as: 0, or 15 through 255 (seconds).

System default: 30 (seconds).

*READ
If READ is specified and the terminal has the 'Receive Interrupt Feature' the system will interrupt a read request to send the assurance signal as long as the user has not entered any data.

Specified as:
Y = interrupt a read.
N = do not interrupt read.

System default: N

Functional Description: The system causes the type ball on the terminal to 'wiggle' by transmitting alternating upper case - lower case shift characters.

The period between the transmitting of the characters is determined by the 'TIME' parameter.

If the TIME is zero, then no transmission occurs.

If the user has specified READ=Y, and the terminal has the correct feature, the system will interrupt a read to send the 'ball wiggle' transmission.

The system will not interrupt a read request if the user has started entering data.

Programming Notes: The blip is supported only on 2741 type terminals. If the user specifies READ=Y for a terminal without the correct feature, the system will attempt the character transmission, but because the terminal will be in the wrong mode, the ball wiggle will not be seen by the user.

Also, any characters entered by the user during this period will be lost.

The system attempts to prevent loss of data by not interrupting a read once the user has started entering data, but because of hardware constraints, there is a very small period of time when the user may enter a character and the system will interrupt to send the 'ball wiggle'; at that time, any data entered is lost.

Example: User enters:

```
BLIP 15
```

The system will blip-wiggle the type ball every 15 seconds as long as there are no other reads or messages to be written to the terminal.

If the user wishes to turn off the ball wiggle, he enters:

```
BLIP 0
```

The system will not wiggle the ball again until the user enters a new BLIP TIME value.

BLIP? Command

BLIP? is used to display the current BLIP settings.

Operation	Operand
BLIP?	

Note: This command has no operands.

Functional Description: BLIP? will display the current BLIP settings in the following format:

```
CURRENT VALUES ARE:
    TIME XX
    READ ACTIVE/NOT ACTIVE
```

The time value is the decimal number of seconds between BLIPS. Read Active specifies that a read will be interrupted to cause the type ball to wiggle.

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: If the user has interrupted a program, BRANCH can be used to resume execution of the program at a different location. BRANCH can also be used as part of a dynamic statement to alter the path of a program.

Cautions: BRANCH should be the last command in a command statement containing more than one command. If not, commands that follow BRANCH are ignored.

BRANCH cannot be used to initiate execution of 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 has stopped execution of his program (which has an ISD). He wants to resume execution at an instruction location labeled with the internal symbol LCCA. He issues the following command:

   User: branch pgm.loca

   The system resumes execution at LOCA.

2. The user wants to alter the execution path of his program (PROG) from location PTA to PTC. He issues the following command:

   User:
   ```
   qualify prog
   at pta; branch ptc
   prog
   ```

   The system passes control to PTC when execution reaches PTA. (See "Program Control" in Section 3 of Part II.)

BUILTIN Command

This command defines an object program (which was written in assembler language) that the user can invoke as a command. (See Section 4 of Part II.)

Operation	Operand
BUILTIN	NAME=command name[,EXTNAME=bpkd macro name] [,PROLIB=data set name]

NAME
designates the name of the command that calls the object program.

Specified as: from one to eight characters, none of which can be embedded blanks, commas, semicolons, equal signs, or apostrophes.

EXTNAME
is 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 between the command and the routine to be called.

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

System default: the value given in NAME is assumed.

PROLIB
specifies the data set in which the BUILTIN is stored.

Specified as: the name of a VPAM data set. If this data set does not exist, it will be created. The BUILTIN is stored in the SYSPRC member of the data set.

System default: USERLIB.

Programming Notes: 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 can be supplied in the object code as part of the PSECT or CSECT 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 User Macro Instructions for a further description. The user can define operands and supply operand values when his user-written command is issued.

Note: If the command BUILTIN is stored in a data set other than USERLIB, the command will not be available until the data set becomes USERLIB or until the BUILTIN command is put in USERLIB.

C, CA, and CB Commands

These commands transfer input control from the user's 1052 Printer-Keyboard to the attached 1056 Card Reader.

Operation	Operand
C	

Operation	Operand
CA	

Operation	Operand
CB	

Note: These commands have no operands.

Functional Description: The C, CA, and CB commands indicate to the system that input will come from the 1056 Card Reader, rather than from the attached 1052 Printer-Keyboard. To use these commands, the user places his card deck in the 1056 Card Reader, and then he issues the appropriate command (C, CA, or CB) from the printer-keyboard. The system reads cards from the card reader until the user presses the ATTENTION key on the terminal or until the system reads a K, KA, or KB card from the card reader. The system then reads further input from the printer-keyboard.

These three commands also control the character set that is used on card input. The definitions are as follows:

C -- transfers control to the card reader: if keyboard mode was KA, CA will be card reader mode; if keyboard mode was KB, CB will be card reader mode.

CA -- transfers control to the card reader; card input is converted from 1057 card-punch code to EBCDIC. This command can be used to change the ALPHABET operand without transferring control.

CB -- transfers control to the card reader; card input is converted from 029 card-punch code to EBCDIC. This command can be used to change the ALPHABET operand without transferring control.

Note: The CA and CB commands set the ALPHABET operand.

Example: The user wants to shift from keyboard input mode to card reader input mode.

User: C

The system reads input from the attached 1056 Card Reader. If the terminal mode was KB, card mode is CB; if the terminal mode was KA, card mode is CA.

CALL Command

This command invokes an object module or a PL/I 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. (FORTRAN users should use only main-program names; otherwise, the results are unpredictable. PL/I users should use only OPTIONS (MAIN) procedure names.)

System default: the last module referenced by the system is called.

module parameters
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. Parameters may only take forms acceptable to PCS. These are as follows:

- A command variable
- A quoted string
- A decimal integer
- A floating-point number
- A hexadecimal string
- A register

Specified as: the parameters, separated by commas, expected by the module. A maximum of five parameters is allowed.

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 control to the module most recently referenced by one of

these commands: PLI, ASM, LNK, FTN, LOAD, UNLOAD, CALL with a specified module name, or an implicit call. 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.

When the called module receives control, register 1 contains a pointer to a parameter list if parameters were entered. This list is preceded by a word containing the number of parameters entered. Each word in the parameter list contains a pointer to the actual parameter entered. Each parameter is preceded by a byte containing the length of the parameter, unless the parameter is a command variable and a register. In this case, no length is given. If a parameter is defaulted (denoted by two successive commas), the corresponding pointer in the parameter list is zero.

Caution: If the module called during 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 by a direct call (see below). A direct call follows the command system symbol-resolution process in which PROCDEFs take precedence over modules. If a module and a PROCDEF have the same name, the PROCDEF 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. When you call a PL/I program to execute, you must use the module name.

Examples:

1. The user wants to compile and execute program MYPRG.

   User: ftn myprg

   The system compiles and stores myprg.

   User: call

   The system invokes MYPRG.

2. The user wants to call module XYZ and to pass five parameters.

   User: call xyz, par1,,,par4

   System: (invokes XYZ and places a pointer to the parameter list in register 1.

3. The user wants to call module XYZ and to 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 execute an object program, he may do so by entering the module name and the operands expected as parameters by the module. The system loads the module (and any implicitly referenced modules) and passes control to the module.

When a PROCDEF and a module have the same name, the PROCDEF is called. The CALL command invokes the module.

When the specified module or PROCDEF cannot be found, a diagnostic message is issued.

When a module expects parameters, all parameters must be specified, including commas for null values. Parameters are passed to the module as described above.

Caution: A direct call is not permitted in a dynamic statement.

Examples:

1. Load and execute module ABC.

   User: abc

   The system invokes ABC.

2. Load module ABC, pass parameters X, Y, and Z, and execute.

   User: abc x,y,z

   The system invokes ABC and places a pointer to the parameter list in register 1.

CANCEL Command

This command eliminates a nonconversational task or job.

Operation	Operand
CANCEL	BSN=batch sequence number

BSN
identifies the nonconversational task to be canceled.

Specified as: a one- to four-digit BSN 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, is printed and includes a message indicating the reason for task termination.

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

The user is informed if the task 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 the nonconversational task (before execution) identified as BSN 1214.

User: execute xyz

System: BSN = 1214

User: cancel tsn=1214

System: CANCEL ACCEPTED

CATALOG Command

This command creates a catalog index for a generation data group or renames a data set.

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

Form 1

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

Form 2

Operation	Operand
CATALOG	GDG=generation data group name,GNO=number of generations [,ACTION={A|O}][,ERASE={Y|N}]

DSNAME
identifies the data set. VAM data sets must be cataloged; physical sequential data sets must be defined by a DEF command within the current task or must be catalged. The data set must reside on a direct access device or on a 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.

ACC
specifies the access qualification for the data set.

Specified as:

R - read-only.
U - unlimited.

Default: U, if the catalog entry is new; otherwise, no change is 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; the maximum number of characters is 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; the maximum value is 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 are to be removed from catalog.
O - only the oldest generation is to be removed.

System default: O.

ERASE

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

Specified as:

Y - old generation data sets to be erased.
N - old generation data sets to be saved.

System default: N.

Functional Description: CATALOG offers these options:

1. Rename a VAM or physical sequential data set (Form 1)
2. Create or alter a catalog entry for a physical sequential data set (Form 1)
3. Create a generation data group (GDG) for VAM or physical sequential data sets (Form 2)

When a data set is renamed, the system changes the data set labels on the direct access volumes containing the data set.

When a physical sequential data set is cataloged, 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 a GDG is created, the system enters the GDG name in the 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 the deleted generations.

A generation of a GDG can be catalcged 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 catalcgs a generation to a GDG and exceeds the maximum number of generations maintained, the system removes all generations or the oldest generaticn, depending on the option selected by the user.

Note: All VAM data sets, both public and private, are cataloged automatically by the system when they are created.

Caution: The user should not rename data sets that reside on magnetic tape volumes; he may lose the data sets if he renames them in his catalog. CATALOG (Form 1) cannot be used to update the entry for a GDG.

Programming Notes: To change the catalog entry for a GDG, use this procedure:

1. Temporarily catalog each member as a separate data set by renaming it. For example:

   ```
   catalog sampleds(0),u,,sampleds1
   ```

2. Delete the GDG by using the DELETE command.

3. Define a new GDG by using the CATALOG command, specifying the new options desired.

4. Add the temporarily cataloged members to the GDG by renaming them the original name. (Use the NEWNAME operand.)

Once a GDG has been cataloged, the user can add generations to the group. When he creates a VAM data set and names it as a generaticn (by appending a generation number) of a GDG, the system catalogs the generation. Uncataloged physical sequential data sets can be added as generations with CATALOG (Form 1). Both VAM and physical sequential data sets that are already cataloged can become generations of a GDG by renaming them with CATALOG (Form 1).

A data set may be renamed as a generation of a generation data group by using the NEWNAME operand (Form 1). The new name must be specified as a generation level, for example, name(+1). If the data set being renamed is the first generation in the group, Form 2 of the CATALOG command must be entered first to create a catalog index for the generation data group.

A new generaticn can be cataloged with either an absolute or relative generation number; any cataloged generaticn can be referenced with either number. When using relative numbers, the user must know the actual generation being referenced. The newest generation has relative generation number 0.

If a private VAM data set is deleted from the catalog, the EVV command, not CATALOG, must be used to reenter the data set in the catalog.

A user who has been granted unlimited sharing access to one or more levels of another user's catalog may add entries to that catalog. When naming such entries, the user must include qualifiers with the same names that he assigned to his SHARE command for that catalog. Similarly, if he wants to rename a shared data set, he may only rename the SHARE qualifier as a part of the new name. The sharer cannot change the owner's catalog.

When catalcging a new physical sequential data set, the user can use the NEWNAME operand to specify a second name, and that name is assigned to

the catalog entry. For example, a data set created under OS or OS/VS may have a name that is too long; with NEWNAME, the user can rename it to suit TSS requirements.

Examples:

1. The user wants to rename data set X.X2 to SIMUL.SK. To get the catalcg entry changed, he enters:

   User: catalog x.x2,u,newname=simul.sk

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

   User: catalog gno=10,gdg=aset

   The system creates a catalog index entry.

3. The user wants to catalog a new generatiCn cf generation data group ASET. It is assumed that the generation has been cataloged:

   User: catalog xgz,u,,aset(+1)

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

4. In a subsequent task, the user wants to catalog anCther new generation cf generation data group ASET. The new generation is assumed to have been cataloged:

   User: catalog abc,u,,aset(+1)

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

5. A user (user2) wants to add the previcusly defined physical sequential 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,user2,u
   User2: share do,user1,ownerds=*all
          ddef ddni,dsname=do.file.b4,disp=new
          catalog do.file.b4,n,u
   ```

   The system creates a catalog entry for DO.FILE.B4 in the catalog for user2.

CB Command

(See C, CA, and CB Commands)

CDD Command

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

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

DSNAME
identifies the cataloged line data set that contains prestored DDEF commands.

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 positionally.

Functional Description: The CDD command retrieves one or more DDEF commands from the specified data set and 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 DDEF commands that contain invalid operands are displayed, as are diagnostic messages issued by DDEF.

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 if any other command appears in the data set. These error lines are ignored and are not printed. The conversational user has the option of either skipping the erroneous records in the data set or canceling the CDD command; a nonconversational task is terminated.

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 that are stored in the 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)

The system processes the DDEF commands in the data set, and then prints information similar to the following:

```
DDEF NOW1,VI,DSNAME=WINDUP,DISP=CLD
DDEF NOW2,VI,DSNAME=GOONNU,DISP=CLD
DDEF NOW3,VS,DSNAME=STAR,DISP=CLD
```

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

User: cdd payroll.p,jbacct

The system processes the specified DDEF command and then prints information similar to the following:

```
DDEF JBACCT,PS,DSNAME=LEAD.T,DCB=(DEN=2),
UNIT=(TA,9),VOLUME=(,043591),LABEL=(2,SL,RETPD=2),
DISP=OLD
```

CDS Command

This command copies a data set or specified members of a partitioned data set.

Operation	Operand
CDS	DSNAME1=input data set name[(member name[,...])], DSNAME2=copy data set name[(member name)] [,ERASE={Y|N}] [,COPYBASE=first line number, COPYINCR=increment] [,REPLACE={R|I}]

DSNAME1
identifies the data set to be copied; VAM data sets must be cataloged; physical sequential data sets 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) member names of a VPAM data set. When specified, the member names are separated by commas and enclosed in parentheses, and they immediately follow the VPAM data set name.

Note: A PS data set can only be copied to another PS data set.

DSNAME2
specifies the data set name to be assigned to the copy of the data set. The data set can be already defined by a DDEF command within the current task. Otherwise, CDS defines it with the same data set organization as DSNAME1.

Specified as: a fully qualified data set name and (optionally) a member name must be of a VPAM data set. The member name must be enclosed in parentheses and must immediately follow the VPAM data set name. When multiple members of the input data set are specified, the data set copy must be a partitioned data set with no member names specified. If not, the CDS command is canceled and the user receives a diagnostic message.

Note: A PS data set can only be copied to another PS data set.

ERASE
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: Y - data set to be erased.
N - data set to be saved.

System default: N.

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 is ignored. If the data set being copied is physical sequential, this operand is ignored.

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

Specified as: from one to seven decimal digits. An all-zero starting line number is invalid.

System default: no renumbering occurs; COPYINCR must also be defaulted.

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

Specified as: from one to seven decimal digits. An all-zero increment is invalid.

System default: 100, when renumbering.

Note: COPYBASE and COPYINCR may only be specified for line data set copies. When COPYBASE and COPYINCR are specified for a line data set copy, the first seven bytes of each record in the copy are the line numbers, and the eighth byte is the origin character. Thus, when a VSAM data set is the source for a line data set copy, the first eight bytes of each source record are overlaid with line numbers. When a line data set is source to a line data set copy, the source record line numbers are overlaid with the new line numbers.

REPLACE
allows the user to specify that duplicate members are replacements or are to be ignored with a suitable diagnostic.

Specified as:

R - replace an existing member in the data set copy with a member from the input data set.
I - ignore any member in the input data set that is duplicated in the data set copy.

System default: R.

Note: When COPYBASE and COPYINCR are specified, this operand is ignored.

Functional Description: The CDS command has two functions. The first function is to merge or overlay members of one partitioned data set with members of another partitioned data set. (The characteristcis of the data sets are presented in Table 16.) This is known as member processing. The user specifies the function with the following restrictions:

1. DSNAME1 and DSNAME2 must be names of virtual partitioned data sets.

2. DSNAME2 has no member name specified with it.

3. DSNAME1 may have one member name, a list of member names, or no member names specified with it.

Member processing causes the specified members of DSNAME1 (if no member name is specified, all members are processed) to be copied into DSNAME2

with duplicate members handled according to the specification for the REPLACE operand. The CCPYBASE and COPYINCR parameters have no meaning in this type of processing.

When CDS does member processing it moves member name aliases and user data along with the data and the member name. The CDS command can be used to copy a program library; all the aliases are preserved. However, if an alias for a member of the input data set is an alias for a member that is not being replaced in the output data set, the copy of that input member is not made.

Table 16. Characteristics of data sets that are used by CDS

Data Set 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	Can be defined by previous DDEF in current task
VI VS VI VS	VI VS VS VI	Must be stored on direct access volume	Must be cataloged	
VS	VS or VI member of VPAM data set	Source data set and VPAM data set receiving member must be on direct access volumes	Must be cataloged	Can be defined by previous DDEF in current task, unless new member of existing cataloged data set
VI	VS or VI member of VPAM data set			
VS member of VPAM data set	VS VI	VPAM data set provides source and copy data set, stored on direct access volumes	VPAM data set must be cataloged	Can be defined by a previous DDEF in the current task
VI member of VPAM data set	VI VS			
VS member of VPAM data set	VS or VI member of VPAM data set	VPAM data sets stored on direct access volumes	VPAM data set must be cataloged	
VI member of VPAM data set	VS or VI member of VPAM data set			

Whenever a member is not copied, because of a duplicated alias or because REPLACE=I, CDS issues a message that contains the name of the member and the reason the member was not copied.

The second function is to copy any data set or member of a data set and make it another data set or member of another data set. (See Table 16.) This function has the following restrictions:

1. DSNAME1 and DSNAME2 may be the names of any type of data set.

2. If DSNAME1 or DSNAME2 are virtual partitioned data sets, only one member name must be specified.

The REPLACE operand has no meaning for this function.

When a starting line number is specified, the lines of the output data set are numbered. The specified or default increment value is used, and the line numbering within the original data set is not affected.

Cautions: If this function is used to create a member of a virtual partitioned data set, no user data or aliases are provided. The CDS command is restricted to data sets on direct access or magnetic tape volumes. CDS cannot be used to change record formats.

A copy of a member of a partitioned data set may have VISAM or VSAM organization.

The user may specify a VISAM organization for a data set copy even though the original data set organization is VSAM. Each record of the data set must contain a key. The user must use a DDEF command to specify the new data set organization (VS), the key length (KEYLEN), the padding (PAD), and the key position (RKP). If the user fails to provide these optional values (except PAD), and his task is conversational, he is prompted for the values. If the task is nonconversational, no copy is made. The PAD operand is optional; and if it is omitted, it is assumed to be zero.

Examples:

1. The user wants to copy the 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. He enters the following command:

   User: `ddef ddnz,vs,dsname=twin.firstl`
   `cds firstl,twin.firstl`

2. The user wants to copy three members, A, B, and C, from LIB1, a VPAM data set, into LIB2, a VPAM data set that has members named A and C. He wishes to replace members A and C and add B. Both data sets are cataloged. The command he enters is:

   User: `cds lib1(a,b,c),lib2,,,,r`

   The system copies members A, B, and C with aliases and user data.

3. The user wants to copy a VISAM member A from LIB1, a VPAM data set, into LIB2, also a VPAM data set, with the name A maintained for the new member in LIB2. He also wishes to renumber A with a base of 50 and an increment of 10. He enters the following command:

   User: `cds libl(A),lib2(A),,copybase=50,copyincr=10`

   The system copies member A with all aliases and user data lost.

4. The user wants to merge the VPAM data sets LIB1 and LIB2. He enters:

   User: cds lib1,lib2,,,,i

   The system copies all members from LIB1 into LIB2 unless a duplicate member name is found in LIB2, in which case that member is ignored. All aliases and user data are copied.

5. The user wants to copy VSAM data set SEQ.DATA, and he wants to make it a VISAM data set named VI.DATA. He wants to use 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. He enters the following commands:

   ```
   User:   ddef dd2,vi,dsname=vi.data,dcb=(recfm=f,lrecl=512,-
           rkp=3,keylen=3,PAD=10)
           cds seq.data,vi.data
   ```

   The system copies the data set.

6. The user has a 9-track tape (volume serial number 000126) that contains BSAM data sets. He wants to copy the third file on the tape onto a scratch tape. The serial number of the scratch tape will be supplied to the system by the operator.

   ```
   User:   ddef tape1,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
   ```

   The system copies the data set.

CHGPASS Command

This command will change, add, or remove your password.

Operation	Operand
CHGPASS	[NEWPASWD=password]

NEWPASWD
specifies your new password

Specified as: from one to eight alphameric or special characters (except tab, comma, backspace, percent sign, equal sign, and left and right parentheses).

System default: if you do not enter the operand, the system overtypes a line that is a prompt for you to enter a password.

Functional Description: CHGPASS adds, changes, or removes your password.

If you enter the command without the operand, the system prints out a message that prompts you for the new password and prints an overtyped area in which you enter the new password. After you enter the new password, the system validates it and prompts you for your current password. You enter your current password in an overtyped area. If the current password is valid, the new password becomes your current password.

If you enter the operand of this command, the specified password becomes your new password after you have successfully entered your current password.

If you wish to nullify a previous password, but do not wish to replace it, enter the command with no operand. When you receive the overtyped line, press the RETURN key on your terminal.

Examples:

1. You wish to enter a new password as an operand:

   User: chgpass passwd2

   The system prompts you to enter your current password by printing a message and an overtyped line.

   User: oldpass

   The system replaces your old password -- oldpass -- with the new password -- passwd2.

2. You wish to enter a new password with password security:

   User: chgpass

   The system prompts you to enter your new password and prints an overtyped line.

   User: passwd3

   The system prompts you for your current password and prints an overtyped line.

   User: oldpass

3. You wish to nullify your password, but you do not wish to replace it with a new password. You enter:

   User: chgpass

   The system prompts you for your new password and prints an overtyped line.

   User: (presses the RETURN key to default the password)

   The system prompts you for your current password and prints an overtyped line.

   User: oldpass

CLOSE Command

This command closes a user's data sets when the normal path of processing is interrupted, either by the system or by the user, and the data set cannot be closed at the program level.

Operation	Operand
CLOSE	[DSNAME=data set name][,TYPE=T] [,DDNAME=data definition name]

DSNAME
the name of the data set to be closed.

Specified as: the fully or partially qualified name of the data set or data sets to be closed.

System default: All user data sets, with the exception of USERLIB, are closed if a DDNAME parameter is not specified.

TYPE
a temporary close (TYPE=T) is to be performed for the user's data sets.

Specified as: T

System default: A normal close is performed.

DDNAME
the data definition name of the data set to be closed or the leading characters of a data definition name that are common to a group of data sets that are to be closed.

Specified as: from one to eight alphameric characters, the first of which is alphabetic, or a quoted string. If the quoted string is not eight characters, it is padded to that length with blanks.

System default: The data set specified by a fully qualified data set name, or all the data sets identified by a partially qualified name, or all user data sets except USERLIB when DSNAME is defaulted are closed.

Functional Description: The system looks for the data sets you specified in the command. If a data set is found and it is a user data set that is not a JOBLIB, it is closed. If the data set cannot be closed, the system issues a message. The system issues a message, also, if the data set cannot be found.

This command should be used when your task or program is abnormally terminated, either by you or by the system. If you are not sure that data sets have been closed, issue the CLOSE command for your data sets. Control is returned to you after CLOSE has been executed; that is, CLOSE does not terminate abnormally.

Programming Notes: This command closes data sets belonging to a user; it cannot be used to close system data sets.

The default of data set name does not cause the user's USERLIB to be closed; USERLIB is closed only when explicitly specified. If both DSNAME and DDNAME are specified and the data set name is partially qualified, only the data set with the specified data definition name is closed.

If the specified data set is a job library (JOBLIB), all DCBs are closed except the system DCB that makes the data set a JOBLIB (USERLIB is defined as a JOBLIB). The RELEASE command can be used to close this DCB and release the JOBLIB.

A group of data sets can be closed with the DDNAME operand by specifying the leading characters of the data definition name that are common to the group (for example, FT for FORTRAN data sets).

Caution: TYPE=T must not be specified for duplexed VAM data sets.

The following conditions occur when the CLOSE command is used and TYPE=T has been specified. (1) PROCDEFs cannot be executed after the command has been executed for USERLIB, and messages in the user's SYSMLF cannot

be issued until the user issues another LOGON for the task. (2) The command closes all open members of a partitioned data set. Therefore, issue a FIND macro instruction for the partitioned data set before continuing processing from a program, or reissue the EDIT command before continuing processing with the text-editing commands.

COBOL Command

This command will invoke the OS/VS COBOL program product using the Program Product Language Interface.

Operation	Operand
COBOL	NAME=modulename [,OSOPTS=(opt1,opt2,...)] [,SOURCEDS=sourcedsname]

NAME
identifies the name by which the object program will be known to TSS. It consists of one to eight alphameric characters, the first of which is alphabetic. If the SOURCEDS option is not specified, there must exist a dataset called SOURCE.name which is assumed to be the source program to be compiled.

OSOPTS
specifies a list of OS/VS options to be in effect during the compilation.

Option	Significant Characters	Option	Significant Characters	Option	Significant Characters
LINECNT	CNT	XREF	XRE	VERB	VER
SEQ	SEQ	BATCH	BAT	ZWE	ZWB
FLAGE(W)	LAG,LAGW	NAME	NAM	ENDJOB	END
SIZE	SIZ	SXREF	SXR	TEST	TES
BUF	BUF	STATE	STA	LVL	LVL
SOURCE	SOU	TERM	TER	ADV	ADV
DECK	DEC	NUM	NUM	COUNT	COU
LOAD	LOA			DUMP	DUM
SPACE	ACE	LIB	LIB	LSTONLY/LSTCOMP	LSTO/LSTC
DMAP	DMA	SYMDMP	SYM	LCOL1/LCOL2	CL1/CL2
PMAP	PMA	OPTIMIZE	OPT	FDECK	FDE
SUPMAP	SUP	SYNTAX	SYN	CDECK	CDE
CLIST	CLI	CSYNTAX	CSY	L132/L120	L13/L12
TRUNC	TRU	RESIDENT	RES	VBSUM	VBS
APOST	APO	DYNAM	DYN	VBREF	VBR
QUOTE	QUO	SYSx	SYS		

The compiler options are as follows:

[SIZE=YYYYYYY] [,BUF=YYYYYY] [,SOURCE|NOSOURCE] [,DMAP|NODMAP]

[,PMAP|NOPMAP] [,SUPMAP|NOSUPMAP] [,LOAD|NOLOAD] [,DECK|NODECK]

[,SEQ|NOSEQ] [,LINECNT=nn] [,TRUNC|NOTRUNC] [,CLIST|NOCLIST]

[,FLAGW|FLAGE] [,QUOTE|APOST] [,SPACE1|SPACE2|SPACE3][,STATE|NOSTATE]

[,XREF|NOXREF] [,SXREF|NOSXREF] [,NAME|NONAME] [,BATCH%NOBATCH]

[,TERM|NOTERM] [,PRINT|NOPRINT[(*|dsname)]] [,SYMDMP|NOSYMDMP]

[,OPTIMIZE|NOOPTIMIZE] [,SYNTAX|NOSYNTAX] [,LVL=A|B|C|D]

[,TEST|NOTEST] [,ENDJOB|NOENDJOB] [,CSYNTAX|NOCSYNTAX]

[,RESIDENT|NORESIDENT] [,DYNAM|NODYNAM] [,VERB|NOVERB] [,ZWB|NOZWB]

[,SYST|SYSx][1] [,ADV|NOADV] [,COUNT|NOCOUNT] [,DUMP|NODUMP]

[,LSTONLY|LSTCOMP|NOLST][2] [,LCCL1|LCCL2][2] [,FDECK|NOFDECK][2]

[,CDECK|NODECK] [L132|L120] [,VBSUM|NOVBSUM] [,VBREF|NOVBREF]

[1]If the information specified contains any special characters, it must be delimited by single quotation marks instead of parentheses. If the only special character contained in the value is a comma, the value may be enclosed in parentheses or quotation marks. The maximum number of characters allowed between the delimiting quotation marks or parentheses is 100.

[2]These options are used to request the lister feature.

Additional information is available in Appendix J, and the OS/VS COBOL Programmer's Guide.

SOURCEDS
specifies the name of the input dataset to be compiled.

CONTEXT Command

This command replaces a string of characters within one line, a range of lines, or all lines in a region or in a 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 line of a range of lines in the current region or data set that is to be searched for STRING1.

Specified as: a one- to seven-digit line number in decimal 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 character position of the specified line, he can specify the starting position as an absolute one- to four-digit decimal number enclosed in parentheses and immediately following tne line number. The first character of text is position 1.

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

N2
identifies the last of a range of lines in the current region or data set that is to be searched for STRING1.

Specified as: a one- to seven-digit decimal 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 character position of 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. This ending character is included in correction processing. The first character of text is position 1.

System default: When N1 is specified, it is the only line searched; otherwise, the entire data set or region is searched from the beginning.

STRING1
designates the character string (called search argument) that is to be searched for within the range N1 to N2. The character string must be located in one line or it will not be found.

Specified as: a normal or quoted string; it may not be null.

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: Wherever 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 is extended to make room for the replacement string; if the replacement string is shorter, the line is processed so that no extra spaces remain in the line after the command is executed.

If STRING1 is not specified, the user is warned, and the command is ignored. If STRING1 and STRING2 are enclosed in apostrophes, the apostrophes are stripped off before execution of CONTEXT. After execution, the CLP is set to the line following the last line processed (N2); the user is prompted for another command. If N2 is the last line in the data set or region, CLP is set to N2 plus the value of INCR.

Caution: A language-processing command (EDIT, PROCDEF, or PLI) must be invoked before the CONTEXT command is issued.

If you use CONTEXT or CORRECT to update a line of a source program that was entered via punched cards, you must maintain punched-card format.

Programming Notes: The CONTEXT command can be used to replace symbols in source language modules. In this use, 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 a region or line data set.

Since CONTEXT does not display the lines in which string replacement has occurred, the user may want to use the LIST command following CONTEXT.

Examples: A data set contains 20 lines, numbered from 100 through 2000.

1. The user wants to replace the string ABCDEF with UVWXYZ. He enters:

User: context ,,abcdef,uvwxyz

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

   User: context 500,1000,abcdef,uvwxyz

   The system searches lines 500 through 1000, replacing ABCDEF with UVWXYZ, and issues:

   _

3. The user wants to replace ABCDEF only in the first 50 character positions of line 1200 (assume CLP=1000). He enters:

   User: context +2,+2(50),abcdef,uvwxyz

4. The user wants to delete ABCDEF. He specifies an explicit null string in the command (assume CLP=1000). He enters:

   User: context -3,last,abcdef
   System: _

5. The user wants to replace in a region the string JOHN'S HOUSE with another string. He enters:

   User: context 0,last,'john''s house','*!@/#$'

   The system searches entire region, replaces JOHN'S HOUSE with *!@/#$, and issues:

   _

CORRECT Command

This command changes characters or inserts characters in one or more lines of the current region or data set.

Operation	Operand
CORRECT	[N1=starting line][,N2=ending line][,SCOL=starting column] [,CORMARK=replacement correction characters][,CHAR={C|M|H}]

N1

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

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

LAST - last line in the current region.

System default: The value of CLP within the region.

N2

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

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

LAST - last line in the current region.

System default: N1 is assumed if specified; otherwise, the value of CLP within the region is assumed.

SCOL

specifies the character position within the text of each line, 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 of data is position 1.

Specified as: from one to four decimal digits.

System default: position 1.

CORMARK

identifies the correction characters that are to replace the standard correction characters. The standard correction characters (*$@%#) are replaced from the left by a direct substitution. Any that are not entered are assumed to be unchanged. All characters up to the one to be changed must be entered.

Specified as: a normal or quoted string.

System default: The standard correction character.

The standard correction characters, or the corresponding replacement correction characters, and their functions, are:

* -- duplicates the character directly above the * and all characters to the right of that character -- until either the next correction character or the end of line is encountered.
$ -- duplicates the character directly above the $. All replacement characters on the correction line are substituted for the corresponding characters on the original line until either the next correction character or end of the original line is encountered.
@ -- duplicates the character directly above the @. If the @ is immediately followed by another correction character or by the end of the line, characters from the replacement line are inserted immediately after the character that is above the @. The rest of the line is moved to the right to make room for the insertion. If the @ is not immediately followed by another correction character or by the end of the line the number of spaces between the @ and the next correction character (or the end of the line, if no other correction character follows the @) mark the characters in the original line that are replaced by the characters in the replacement line.
% -- removes the character directly above the %. All characters to the right of the character above the % are shifted left one position, and all other characters to the right of that character are duplicated until either the next correction character or the end of line is encountered.
# -- functions as does the @, except that hexadecimal characters are inserted and this symbol cannot be the last character in the line before the carriage return. (See "Caution," below.)

CHAR

indicates the type of input expected.

Specified as:

C - character (the line is displayed in character notation)
H - hexadecimal (the line is displayed in hexadecimal)
M - mixed (functions the same as CHAR = H for input and display of the line)

System default: C.

Note: If CHAR=H and the user enters nonhexadecimal data, the CORRECT command is canceled. Also, if CHAR=H, hexadecimal data is expected following the @ and the # characters, and the $ character may input hexadecimal data in the correction line. If CHAR=C the # character can be used to enter hexadecimal replacement characters.

Functional Description: When one line is specified to be corrected, the system displays that line. You enter a correction line (a line that contains the correction or the correction characters, or both). Until the system detects a correction character in the correction line, it replaces the characters in the original line with those given in the correction line. The system prompts you for a replacement line if the @ or the # characters are used to indicate replacement. If LIMEN is defaulted to I, the system prompts with a message for you to enter the replacement line; otherwise, only the keyboard is unlocked to indicate that the replacement line can be entered.

An end of line in a field marked by *, $, or % within the correction line causes the remainder of the original line to be duplicated. An end of line in a field marked by $ terminates the line.

Following execution of the CORRECT command, the CLP is set to the next line after N2 (or N2 plus the value of INCR if N2 is the last line), and the user is prompted for a command.

Caution: Unprintable EBCDIC characters in the text appear as spaces when the line to be corrected is displayed unless CHAR=M or CHAR=H is specified. A language-processing command (EDIT, PROCDEF, or PLI) must be invoked before the command is entered.

You can make a record longer by inserting data (use the @ correction character, for example). You cannot add data to the end of a record. If you do, the system cancels the operation and prints a message.

If the # correction character is the last character in the line before the carriage return, the line is canceled.

If you use CONTEXT or CCRRECT to update a line from a source data set that was entered via punched cards, you must maintain punch-card format on that line. (Column 72 is still used for the continuation character.)

Examples: In the following examples, the user enters the CORRECT command, specifying the line to be corrected. The system displays the line, as it exists. The user enters the correction line (the third line in the examples) and a replacement line (fifth line), if one is required.

1.
```
User:       correct 400
System:     STEMS3660
User:       @* $ *%
System:     ENTER REPLACEMENT LINE
User:       ys
Sys,User:   list 400
System:     0000400 SYSTEM 360
            CLP SET TO 0000500
```

In the example the @ in the correction line caused the YS from the replacement line to be inserted after the character that appears

above the @ in the original line; the $ followed by a blank caused the blank to replace the character above it (S); and the % caused the character above it (6) to be deleted and all following characters to be shifted to the left. The characters above the * were duplicated until the next correction character was found.

2.
```
User:       correct 104
System:     COMPVTE X1
User:       *  $u*
Sys,User:   list 104
System:     0000104 COMPUTE X1
            CLP SET TO 0000105
```

In the example the $ followed by the U caused the U to replace the character above the $.

3.
```
User:       correct 27
System:     CONTNUE
User:       *  @
System:     ENTER REPLACEMENT LINE
User:       i
Sys,User:   list 27
System:     0000027 CONTINUE
            CLP SET TO 0000028
```

Again, in this example the @ is used to indicate an insertion after the T.

4.
```
User:       correct 15
System:     XYZ 1345 CC MPTE X
User:       abc@     *  % @
System:     ENTER REPLACEMENT LINE
User:       1 3,4,5@u
Sys,User:   list 15
System:     0000015 ABC L 3,4,5 COMPUTE X
            CLP SET TO 0000016
```

In the example XYZ is replaced by ABC; the four blanks immediately after the @ in the correction line indicate that the characters in the replacement line are to replace the characters above the blanks; the blank between the O and M is deleted; and the U is inserted.

In the following example, the user has a data set named PARTS. Positions 15-18 of lines 300-800 contain a year that is incorrect. He issues the following sequence of commands to get the errors corrected.

```
User:       list 300(15),800(18)
System:     1966
            1966
            1956
            1966
            1866
            k866
            CLP SET TO 0000900
User:       correct 300,800
```

(Note: the system does not print out any data. The keyboard is unlocked and the user enters his correction, using the conventions described above.)

```
User:       *           $1968*
```

DATA Command

This command creates either a line data set or a VSAM data set.

Operation	Operand
DATA	DSNAME=data set name ,RTYPE={I|LINE|FTN|CARD|S} [,DBASE=first line number] [,DINCR=increment]

DSNAME
identifies a data set or a 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 must immediately follow the VPAM data set name.

RTYPE
indicates the organization of the data set specified.

Specified as:

I - line data set organization is required.
LINE - same as above.

FTN - line data set organization is required and the input is a FORTRAN source data set in punch-card format. The card format is converted to keyboard format with keyboard continuation conventions as it is placed into the data set. The resultant data set may be updated from a terminal without any special consideration being required for multicard statements. Trailing blanks are stripped from statements that are not continued.

CARD - a VSAM fixed-length data set is created (no line numbering). The record length is 80 characters. Normally, this option is specified when the user is creating a data set for FORTRAN data from card input. It may also be specified to build a data set from the keyboard conversationally. In this case, leading blanks are not stripped off, and all input goes into the data set in the form it is entered from the terminal. If the line entered is less than 80 characters, it is padded with blanks to create a record that is 80 characters long. If the record entered is greater than 80 characters, it is truncated to 80 characters.

S - a VSAM variable-length data set is required (this generates a record preceded by a four-byte length field) and a one-byte origin field (keyboard/card reader indicator).

System default: S.

DBASE
identifies the starting line number of the line data set being created.

Specified as: from three to seven decimal digits, the last two of which must be zeros. An all-zero starting line number is invalid. (See "Note" under DINCR.)

System default: 100.

DINCR
specifies the value by which the line numbers in the data set are to be incremented.

Specified as: from three to seven decimal digits, the last two of which must be zeros. An all-zero increment is invalid.

System default: 100.

Note: The specification of DBASE and DINCR is invalid for a sequential data set (indicated by defaulting RTYPE).

Functional Description: Either a line data set is selected or a VSAM data set is created. The user can modify, correct, insert, and delete lines only in a line data set.

If the user's task is conversational and LINENO=Y, the DATA command prompts for entry of data. If indexing was specified, the system requests each line by issuing the current line number; if indexing was not specified, the system prompts for each line by issuing a pound sign (#). When the user does one of the following,

1. Enters %E
2. Enters a single break character as the first character of a line
3. Presses the ATTENTION key

the data set is closed and command mode resumed. (If the ATTENTION key is pressed, the data on the current line is not entered into the data set.) In each case, the system prompts the user for his next command. The user may then reopen the data set and continue to build it or a member of it by issuing another DATA command.

If an old data set is specified, the user is prompted with the first line number after the end of the data set. If the old data set is VISAM, the user is prompted with the last line in the data set plus the increment value. If the user specifies his own base value, he is prompted with the line number specified.

Lines being entered for a line data set can be modified, corrected, or deleted, and new lines can be inserted by following the conventions listed below. (See also "Language Processing" in Section 3 of Part II.)

1. To modify or correct a line of a line data set, enter:

   %line number,data

   where:

   line number
   identifies the line to be replaced by a modified or correct line.

   data
   is the replacement line of is data.

2. To insert a new line into a line data set, enter:

   %line number,data

   where:

   line number
   identifies the new line to be inserted. It may be any one- to

seven-digit integer, the value of which specifies the location of the new line within the data set. This value must not exceed the last existing line number.

3. To delete a line or a series of lines from a line data set, enter:

   %D,line number[,last line number]

   where:

   line number
   identifies the last line to be deleted. If a sequence of lines is being deleted, "last line number" must be higher in value than "line number."

Cautions: When other DATA and MODIFY commands are entered as part of the data set, they must be preceded by multiple break characters because the system closes the data set and immediately executes any command following a single break character. The use of multiple break characters in the data mode is the same as described in Section 4 of Part II. The number of break characters used depends upon the level of nesting.

Programming Notes: The maximum line length is 120 characters of text (not counting the line number) for either a line data set or a VSAM data set with variable record length. For a VSAM data set with fixed-length records, the maximum record length is 128 characters. When records are being entered via the IBM 1056 Card Reader with the AUTO EOB switch on, the maximum record length is 80 characters; and with the switch off, the maximum length is 79.

When a line is being continued the continuation character (a hyphen) is not included in the record placed in the data set. Each line that continues the statement initiated in a preceding line is accepted as if it were 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.

Examples:

1. The user is attempting to construct a line data set named ROVER1.

```
User:      data rover1,line,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:    INVALID CORRECTION NO. LINE IGNORED.
User:      700%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=1,3
           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 that is made up of a sequence of commands. The data set is named COMSET and is tc be used in a BACK command.

```
User:      data comset
Sys,User:  #ftn raader,n,,,y,y,y,y
             .
             .
             .
           #logoff
           #_back comset
System:    BSN=0310
```

3. The user wants to add to two members of a VPAM data set that is named OVAL. One member, CIRCIE, has a virtual indexed sequential organization, and the last line currently in the data set is 500. The other member, SQUARE, has a virtual sequential organization.

```
User:      data oval(circle),line,600,100
```

The system infcrms the user that this is an old VISAM member and prompts him with line number 600.

```
           600 new line of data
           700 another line of data

User:      _data oval(square)
System:    (The user will be adding to the end of the data set.)
                      .
                      .
                      .
```

4. The user wants to create a VISAM data set to execute three assemblies nonconversationally.

```
User:      data assembly,line
Sys,User:  100 logon
           200 procdef assm
           300 param $1
           400 ddef a,vi,asmac
           500 ddef v,vs,asind
           600 asm$1,y,macrolib=(a,v)
           700 print list.$1,,,edit,,,,,accept
           800 __end
           900 assm prog1; assm prog2; assm prog3
          1000 logoff
          1100 _execute assembly
System:    BSN=0489
```

DDEF Command

This command defines a data set and describes its characteristics to the system.

Note: This description does not contain all operands for the command. A complete description appears in Appendix E. The DDEF command is shown below in its expected normal form for typical new public VAM data sets. In this use, most or all of the other operands are defaulted.

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

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

Specified as: from 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.

Default: the value assigned at system generation if the data set is new; the existing organization if the data set is cataloged.

DSNAME
specifies the name by which the data set will be (or is) cataloged and referred to during the current task.

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.

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 (the DCB) is the data definition name. The entry containing the data set definition is maintained until the task is concluded or until the definition is deleted via the RELEASE command.

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

When a data set is defined, and the RET operand (see Appendix E) is defaulted, DDEF assumes that the data set resides on permanent storage with unlimited access. The RET command can be used to change these attributes once they are established.

Cautions: If a user's program is executing in conversational mode and refers to an undefined data definition name, a diagnostic message is issued and the user is prompted for information. In nonconversational mode, the task is abnormally terminated. A FORTRAN user can default his terminal as the "undefined" data source or destination (SYSIN or SYSOUT).

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

Programming Note: 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 the data definition name and the data set name, and all other parameters are ignored. If the user wants to change the other parameters, he must issue a RELEASE command to delete the previously issued DDEF command and re-issue a DDEF command to establish a new system entry for the data set definition.

The DDEF command cannot be used to change the catalog attributes of an existing data set; only the DDNAME can be changed.

Examples:

1. The user wants to define a new VPAM data set. He enters:

   User: ddef ddn1,vp,dsname=group(mem1)

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

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

   User: ddef ddn2,vi,t.trip3

DDNAME? Command

This command allows a user to list the data definition names (DDNAMEs) and associated data set names (DSNAMEs) that are currently defined for the user's task.

Operation	Operand
DDNAME?	[JOBLIB={Y|N}]

JOBLIB
controls which data definition names are displayed at the user's terminal.

Specified as:

N - all currently defined DDNAMES are displayed.
Y - JOBLIB DDNAMEs are displayed in the order in which they will be searched.

System default: N.

Functional Description: Used to review the user's DDNAMEs, with the option of reviewing just JCBLIB DDNAMEs. Along with the DDNAMEs, the associated DSNAMEs are listed. If no operand is entered, all data definition names and corresponding data set names defined in the user's task are listed.

Example:

```
User:     ddname?
System:    DDNAME    DSNAME
          SYSULIB    USERLIB
          LPCMNDX    MACNDX.G0005V00
          LPCMSRC    SYSMAC.G0005V00
          SYSUSE     SYSUSE
          SYSLIB     SYSLIB.G0007V00
          SYSUCAT    USERCAT
          SYSSVCT    SYSSVCT
          SYSCAT     TSS*****.SYSCAT
          SYSOUT     CARDTR
          SYSIN      CARDTR
```

```
User:    ddname? joblib=y
          DDNAME   DSNAME
         SYSULIB   USERLIB
         SYSLIB    SYSLIB.G0007V00
```

DEFAULT Command

This command changes operand default values.

Operation	Operand
DEFAULT	{operand name=[value]}[,...]

operand name
: designates the operand whose default value the user wants to alter or establish.

  Specified as: an operand name from a command or an implicit operand name.

value
: specifies the value to be assumed whenever the specified operand name 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 value that was 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 deleted.

Functional Description: The system adds, replaces, or deletes 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 uses the value he assigns for the remainder of the current task, unless the task profile is made permanent with the PROFILE command. (See also the description of the PROFILE command and Section 6 in Part II.)

Programming Notes: The DEFAULT command can be used to delete previously defined default entries. The user enters the DEFAULT command and specifies a null string to be assigned to the operand name he wants to delete. Also, since some operands have the same value during one task or during successive tasks, 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.

Example: The user wants to change the system default LIMEN=W to LIMEN=I permanently in his user profile. He enters:

```
User:       default LIMEN=I
Sys,User:   profile
```

DELETE Command

This command deletes a data set entry from the user's catalog.

Operation	Operand
DELETE	[DSNAME=data set name]

DSNAME
identifies the cataloged data set that resides on a private volume, or the shared data set owned by another user that 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.

System default: a partially qualified data set name; the user's user identification is the only qualifier.

Functional Description: When the data set name specified is partially qualified, the action of the DELETE command depends on the mode of operation. In conversational mode, DELETE tests the DEPROMPT operand in the user profile to determine whether each fully qualified data set name referenced by the input name is presented to the user. When DEPROMPT=Y, the user is presented one fully qualified data set name at a time for disposition. He responds with D and the data set entry is deleted from the catalog (if the data set is private), or he responds with R (for retain) and no action is taken, or he responds with A (for ALL) to delete all data sets with the partially qualified name. When DEPROMPT=N, all data sets with the partially qualified name are deleted without prompting. In nonconversational mode, the names of all private data sets referenced by the input name are deleted, regardless of the value of DEPROMPT.

When a private input name is fully qualified, a private data set is deleted, regardless of the value of DEPROMPT or the mode of operation.

If the user wants to delete a data set that he has shared, he must specify the same name that he used in the associated SHARE command. For example, if the user issues

```
share my,owner1,*all
```

and has in his catalog MY.ONE, MY.TWO, and MY.THREE, he can only delete at the "MY." qualification level. He cannot delete the catalog entries by issuing:

```
delete my.one
```

Rather, he must issue

```
delete my
```

to delete the catalog entries.

When the input name cannot be found, the command is ignored.

Whenever the user attempts to delete one of his own public data sets, a diagnostic message is issued.

Caution: A VAM data set that has been deleted from the catalog cannot be referenced by the user until he reenters the data set into the catalog (with EVV or SHARE).

If the user tries to delete a shared data set for which a bulk I/O operation is pending, the DELETE command is canceled.

Programming Notes: Initially, DEPROMPT=Y. To change this value, use the DEFAULT Command. (See also Section 6 of Part II and the ERASE Command description in this part.)

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

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

Examples:

1. A conversational user wants to delete the data sets with the partially qualified name A.B. DEPROMPT=Y is in his user profile.

   User: delete dsname=a.b

   The system asks the user to enter D for delete, R for retain, or A for all. It prompts with fully qualified data set name.

   System: A.B.C
   User: r
   System: A.B.D
   User: d
   System: A.B.E
   User: r

   No more data sets begin with A.B so the system prompts for the next command.

2. A conversational user wants to delete all private data sets whose names begin with E.F. He issues the following commands:

   User: default deprompt=n
   delete e.f

DISABLE, ENABLE, POST, and STET Commands

These commands allow the user to control changes to a data set.

Operation	Operand
DISABLE	

Operation	Operand
ENABLE	

Operation	Operand
POST	

Operation	Operand
STET	

Note: These commands have no operands and are ignored when TRANTAB=N.

Caution: A language-processing command (EDIT, PROCDEF, or PLI) must be invoked before any of these commands is entered.

Functional Description: The TRANTAB operand in the user profile may be set to either N or Y. (The system-supplied default value is N.) If the user wants to undo a change made by a text-editing command to a data set when TRANTAB=N, he must do so explicitly. For example, assume that the following line is line 200 of the data set ABELINC that is being edited,

```
0000200 FOUR SCORE AND SEVEN YEARS AGO
```

and the user issues a REVISE command to modify it:

```
revise 200
0000200 eighty-seven years ago
```

If the user decides that he prefers the original wording of the line and wants to restore it, he has to do so explicitly by issuing another command, such as REVISE. Furthermore, if the user has modified a number of lines and wants to undo many of the changes, he has to make each change explicitly.

A facility for simplifying such modifications can be introduced by defaulting TRANTAB=Y. When this option is exercised, the text editor maintains a transaction table in which changes to a data set are recorded. Additions to the data set are noted in one part of this table, deletions in another. The only text-editing commands that do not change lines of the data set, and therefore do not result in entries in the transaction table, are DISABLE, ENABLE, LIST, LOCATE, and POST.

When TRANTAB=Y, the text editor functions in either of two states, disabled or enabled. The text editor is in the disabled state when it is invoked; it may be enabled with the ENAELE command and returned to the disabled state with the DISABLE command.

In the disabled state, records of all changes made to lines since the text editor was last invoked, or explicitly disabled, are maintained in the transaction table. If a modification is made to a line for which a previous transaction table entry exists, however, the existing entry is overlaid by the new one. For example, if the text editor is invoked and used for the existing data set ABELINC as shown in the sequence:

```
User:       default trantab=y
Sys,User:   edit abelinc
Sys,User:   list 200
System:     0000200 FOUR SCORE AND SEVEN YEARS AGO
Sys,User:   revise 200
            0000200 eighty-seven years ago
```

the transaction table has

```
0000200 EIGHTY-SEVEN YEARS AGO
```

as an addition, and

```
0000200 FOUR SCORE AND SEVEN YEARS AGO
```

as a deletion. If the user continues

```
Sys,User:   revise 200
            0000200 a long time ago
```

the transaction table contains

```
0000200 A LONG TIME AGO
```

as an addition, and

```
0000200 EIGHTY-SEVEN YEARS AGO
```

as a deletion. The original wording of line 200, FOUR SCORE AND SEVEN YEARS AGO, is no longer retained in the transaction table.

In the enabled state, the text editor removes entries caused by previous commands from the transaction table as each new command is executed. Consequently, only the changes resulting from the most recently issued editing command that affected a line of data can be found in the transaction table. The text editor becomes enabled when the first editing command that alters a line of data is entered following execution of the ENABLE command. For example, in the following sequence

```
Sys,User:  number 50,100
           enable
           excise 100
```

the text editor does not become enabled until the EXCISE command is executed. If a nonmodifying command, such as LIST, had been inserted between the ENABLE and EXCISE commands, it would have been executed in the disabled state. After the EXCISE command was executed, however, the text editor remained enabled until a DISABLE command was issued, or until the editing session was terminated by a break character followed by an END, PROCDEF, PLI, or EDIT command.

Similarly, the DISABLE command does not disable the text editor until a line-modifying editing command is executed.

The STET command nullifies the changes to the data set that were recorded in the table. For example, in the case of the modification to line 200 of the data set illustrated above, the last modification could have been nullified with the STET command. The sequence is as shown below. (Note that line 300 exists so the system prompts for a command after line 200 has been revised. If line 300 did not exist, the system would prompt for data by printing 0000300.)

```
Sys,User:  revise 200
           0000200 a long time ago
Sys,User:  stet
Sys,User:  list 200
System:    0000200 EIGHTY-SEVEN YEARS AGO
```

Note, however, that the original wording of line 200, FOUR SCORE AND SEVEN YEARS AGO, is lost. A second STET command at this point would only reverse the transaction table once again and return the wording of line 200 to A LONG TIME AGO. Thus, the effect of two consecutive STET commands (or of two STET commands separated only by non-modifying commands such as LIST) is to revert the transaction table to its status prior to issuing the first STET command. STET does not change the disable/enable state of the text editor.

The POST command makes the temporary changes to the data set permanent and permits the user to continue to make temporary modifications. For example, in the following sequence:

```
default trantab=y
edit jbm
0000100 the quick brown fix
0000200 jumped over the lazy dog
```

```
0000300 _context 100,,fix,fox
post
context 200,,dog,dig
stet
list
0000100 THE QUICK BROWN FOX
0000200 JUMPED OVER THE LAZY DOG
```

Note that the STET command only reverses the effect of the second CONTEXT command, since the temporary change created by the first was made permanent by the POST command. POST does not change the disable/enable state of the text editor.

To help explain the effect of the DISABLE, ENABLE, POST, and STET commands, the nature of transaction table entries is illustrated below.

Transaction Table Entries: The examples below describe changes to a data set and entries in the transaction table. On the left are the commands that change the data set; on the right are the resulting entries in the transaction table.

Case 1: A new data set is being created. The text editor is disabled. The sequence is as follows:

```
Sys,User: region name
          0000100 abcdef
          0000200 12345
          0000300 e th
          0000400 _list 200
```

Transaction Table

Additions	Deletions
100 ABCDEF	
200 12345	
300 E TH	

Notice that the LIST command does not change data and therefore does not affect the transaction table. Now assume that a STET command is issued. The table described in Case 1 appears as:

```
Sys,User: stet
```

Transaction Table

Additions	Deletions
	100 ABCDEF
	200 12345
	300 E TH

and lines 100 through 300 no longer exist in the data set.

Case 2: Whenever a change occurs to a line for which a previous transaction table entry exists, the existing entry is overlaid by the new one, no matter what the disable/enable status of the text editor is. Assume the first transaction table and data set from Case 1. If line 100 is changed to ABCDMM, the transaction table becomes:

```
Sys,User: update
User:     100 abcdmm
```

Transaction Table

Additions	Deletions
200 12345	100 ABCDEF
300 E TH	
100 ABCDMM	

Notice that the previous entry for line 100 in the additions column is overlaid and that a change to two characters is treated as a change to the entire line. Now assume that when the system unlocks the keyboard the user types in a break character and the STET command:

User: _stet

Transaction Table

Additions	Deletions
100 ABCDEF	200 12345 300 E TH 100 ABCDMM

Lines 200 and 300 are blank; line 100 is ABCDEF. If a POST command is issued, the transaction table is emptied, but the lines of the data set remain the same. Thus, POST makes the changes irreversible.

Case 3: Now assume that the transaction table appears as in Case 2.

Sys,User: revise 100
0000100 BBCDMM

Transaction Table

Additions	Deletions
200 12345 300 E TH 100 BBCDMM	100 ABCDMM

If STET were issued now, line 100 would become ABCDMM, and not ABCDEF. Hence, when multiple changes are made to a line, only the changes still reflected in the transaction table are reversible. Following STET, the table in Case 3 becomes:

Sys,User: stet

Transaction Table

Additions	Deletions
100 ABCDMM	200 12345 300 E TH 100 BBCDMM

Case 4: When the text editor is enabled, each new table entry overlays existing entries. These line changes result in one table entry.

Sys,User: update
User: 300 xyzabc
_context 200,300,za,by

Transaction Table

Additions	Deletions
300 XYBYBC	300 XYZABC

Only the change made by the CONTEXT command appears in the transaction table. STET issued now could change only line 300.

Now assume an INSERT command is issued, followed by three lines of data for lines 400, 500, and 600, which do not exist. The transaction table contains:

Sys,User: insert 300
0000400 hijkl
0000500 mnopq
0000600 rstuv

Transaction Table

Additions	Deletions
400 HIJKL	
500 MNOPQ	
600 TSTUV	

Three entries appear in the transaction table, since only one execution of the INSERT command was involved. A STET command entered now deletes lines 400, 500, and 600. Notice, however, what occurs if one STET command is followed immediately by another. In the example above, lines 400, 500, and 600, which were moved to the deletions column by the first STET command, would be moved back to the additions column by the second STET command. The effect of a STET command can be reversed with another STET command.

Case 5: The ENABLE command does not affect the transaction table until after the ensuing editing command that changes a line is executed. Assume the editor is disabled and this transaction table:

Transaction Table

Additions	Deletions
100 1 KRST	
200 30405	200 3040506
300 NT476	300 ATXYZ

When ENABLE is issued, the table does not change. If ENABLE is followed immediately by STET (or by LIST or LOCATE and then by STET), the entries are reversed as if the editor is disabled. If, however, ENABLE is followed by:

Sys,User: context 100,300,kr,pq

the transaction table becomes:

Transaction Table

Additions	Deletions
100 1 PQST	100 1 KRST

Lines 200 and 300 of the data set are unchanged.

Cautions: ENABLE and DISABLE do not affect the transaction table until a command that alters a line is executed. Thus, STET should not follow ENABLE immediately because the result is unpredictable.

When multiple changes are made to one line, each succeeding change is entered in the transaction table, overlaying previous entries for that line.

Programming Notes: Since the text editor is normally disabled when TRANTAB=Y, revisions to a data set are temporary. The user can nullify changes with the STET command. When the editor is enabled, revisions are permanent, since only the last change is revocable with STET. POST allows the user to make temporary changes permanent and then continue to make temporary changes.

Following execution of DISABLE, ENABLE, POST, and STET, the user is always prompted to enter a command.

POST and STET do not affect the disable/enable status. DISABLE, ENABLE, POST, and STET do not change the value of the CLP.

Examples:

1. The user issues the following sequence of commands:

```
Sys,User:  default trantab=y,regsize=8
           edit myprog
           region abc
           0000100 data line one
           0000200 data line two
           0000300 data line three
           0000400 data line four
           0000500 _excise 400
           excerpt your prog,pgr,500,700
           context 100,300,line,number
           enable
           number 100,last
           list
           end
```

   In this sequence of commands, the results of EXCISE, EXCERPT, and CONTEXT are made to the data set and recorded in the transaction table. When the NUMBER command which follows ENABLE is executed, these table entries are removed, and the entry from NUMBER exists alone in the table.

2. Assume a STET command is issued between CONTEXT and ENABLE; the effect of EXCISE, EXCERPT, and CONTEXT are canceled.

3. STET appears between NUMBER and LIST; only the effect of NUMBER is canceled.

4. POST appears between EXCERPT and CONTEXT, and STET is issued following CONTEXT; only the effect of CONTEXT is canceled.

5. STET appears between ENABLE and NUMBER; the effect of EXCISE, EXCERPT, and CONTEXT are nullified.

6. STET appears between LIST and END; the effect of NUMBER is canceled.

DISPLAY Command

This command prints the contents and names of specified data fields or expressions on SYSOUT.

Operation	Operand
DISPLAY	data field name or expression[,...] id? data field name or expression[,...]

data field name or expression
specifies the data field or expression to display.

Specified as: an absolute address, the name of a data location, an array, a control section, a symbolic range, an arithmetic or logical expression, a quoted string, or a command variable.

id? data field name or expression
specifies one or more data fields or expressions for which the CSECT name, load address, and length are to be displayed.

Specified as: the characters "ID?" followed by the name of a data location, an array, a symbolic range, or an internal or external symbol. (See "Types of Operand Specification" and "Operand Definitions" in Section 3 of Part II for explanations of these terms.) For multiple requests in one DISPLAY command, the characters "ID?" must be repeated for each data field or expression. Data fields or expressions preceeded by ID? and those not preceeded by ID? are interchangable in one command.

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.

When a quoted string is specified as an operand, the string (without the delimiting quotes) is displayed.

When an arithmetic expression is specified, it is evaluated, and the result is displayed.

The ID? form of the DISPLAY command displays the CSECT name, load address, and length for the specified data location, array, or internal or external symbol. This information is displayed as: ID=csect name LOADED AT address, length BYTES LONG. Multiple requests in one DISPLAY command require that each address or name must be prefixed by the characters ID? (see example 7).

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 'register 6',6e
System:   REGISTER 6
System:   6E = .27182818E + 01
```

2. The user has a 5 by 5 integer array and wants to display the first 10 elements of the array, and the element ARRAY (I,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.

3. The user wants to display the result of 9 + 5.

```
User:     display 9+5
System:   14
```

4. The user wants to display the contents of the data field PATTERN.

```
User:     display pattern
System:   PATTERN=4020202020202020
```

5. The user wants to display the address of the data field PATTERN.

```
User:     display a'pattern'
System:   0042D320
```

6. The user wants to display the CSECT name for symbol JOE. He issues:

```
User:     display id? joe
```

7. The user wants information displayed for symbols JOE, JACK, and SAM. He issues:

```
User:     display id? joe,id? jack,id? sam
```

DMPRST Command

This command performs a time-shared dump and restore of a VAM2 disk volume.

Operation	Operands
DMPRST	FROMDEV={2311|2314|24xx|3330|333B},FRVOLID={volid|(volid[,volid])},TODEV={2311|2314|24xx|3330|333B} [,TOVOLID={volid|(volid[,volid])|PRIVATE}] [,NEWVLID=volid][,WRITCHK={YES|NO}][,LABEL={RETAIN|NO}] [,],RUNMODE={BACK|FORE}

FROMDEV
specifies the device type that the from-volume is to be mounted on. If not given, the command is canceled.

Specified as:

2311 - disk in VAM2 format.
2314 - disk in VAM2 format.
24xx - 9-track dump tapes containing a properly formatted dump.
3330 - 3330-1 disk in VAM2 format.
333B - 3330-11 disk in VAM2 format.

FRVOLID
specifies the volume identification number (VOLID) of each from-volume. If not given, the command is canceled.

Specified as: one to six alphameric characters.

Notes:

1. Only one VOLID may be specified for disk.

2. One or two VOLIDs may be specified for tape, each of which must have a standard label.

3. Multiple tape volids must be specified in the order in which the tapes are to be used.

4. Duplicate VOLIDs for the same device type are not permitted.

5. Blanks contiguous to a comma or parenthesis are ignored.

TODEV

specifies the device type that the to-volume is to be mounted on. If not given, the command is canceled.

Specified as:

2311 - disk in VAM2 format.
2314 - disk in VAM2 format.
2400 - labeled 9-track scratch tapes.
3330 - 3330-1 disk in VAM2 format.
333B - 3330-11 disk in VAM2 format.

Note: the following chart indicates valid FROMDEV and TODEV pairs:

FROMDEV	TODEV
24xx	2311, 2314, 3330, 333B
2311	24xx, 2311, 2314, 3330, 333B
2314	24xx, 2314, 3330, 333B
3330	24xx, 3330, 333B
333B	24xx,333B

24xx legend

2408 -- 800bpi

2416 -- 1600 bpi

2462 -- 6250 bpi

When FROMDEV is a 24xx, TODEV must have a capacity equal to or larger than the disk used to create the 24xx.

TOVOLID

specifies the volume identification of each to-volume.

Specified as: from one to six alphameric characters or PRIVATE.

PRIVATE - a scratch volume is requested.

System default: PRIVATE.

Note: If TODEV=24xx, and more tapes are required than are specified, the system requests scratch volumes.

NEWVLID
specifies the volume identification number to be put in the to-disk label at the completion of the job. This parameter is ignored if TODEV=24xx.

Specified as: from one to six alphameric characters.

System default: the volume identification number in the volume label used (see LABEL) is unchanged.

Note: The NEWVLID may duplicate one already in use.

WRITCHK
specifies whether writing of pages to disk is followed by a read-after-write validity check. This parameter is ignored if the TODEV=24xx.

Specified as:

YES - write checking is performed.
NO - normal error recovery is used.

System default: NC.

LABEL
specifies whether the volume label on the to-disk is to remain, or the volume label on the from-disk is to be used. This parameter is ignored if TODEV=24xx.

Specified as:

RETAIN - use the label on the to-disk.
NO - use the label on the from-disk.

System default: NC.

[,]
necessary, when RUNMODE is specified in positional notation, to maintain system compatibility.

RUNMODE
specifies whether a nonconversational task is to be created to run the dump and restore. This parameter is ignored if the task is nonconversational. If not given for a conversational task, the command is canceled.

Specified as:

BACK - a nonconversational task is created to run the dump and restore.
FORE - the dump and restore is run in the user's conversational task.

Functional Description: DMPRST can be used to dump a VAM2 disk to either a 9-track tape with standard TSS labels or to a VAM2 disk that has been prepared by DASDI. The command can also be used to restore a dump tape to a VAM2 disk. The dump tape must be either one dumped by DMPRST or a standard labeled tape dumped by the independent utility program (DASDDR). A disk dumped to tape with DMPRST may also be restored with DASDDR. If error pages were found and assigned when the disk was prepared by DASDI, pages that fall on error pages are relocated (up to 96 relocations). Otherwise, all data pages are put in their original places.

The DMPRST tape format differs slightly from the independent utility program tape.

1. DMPRST does not dump or restore disk user-labels. If these records are desired, the independent utility Dump/Restore is required.

2. DMPRST, when dumping to tape, writes the IPL records from constant areas and, when restoring, skips these records. If a disk is to be used for IPL, the proper IPL text must be on the to-disk before restoring. DASDI can be used to write this IPL text.

3. If a dump requires more than two tape volumes, this command cannot be used for the restore. The independent utility Dump/Restore can restore a dump of one or more tapes.

All space is assumed to be available on the to-volume. Disk pages not actually used are made available and are left unchanged. No entries are made to or deleted from the catalog.

If RUNMODE=BACK is specified, the maximum command string length (not counting keywords) is 120 bytes.

The tape DDEF uses DSNAME=DR.VAM2.DISK.DUMP and DDNAME=TSU52814. These names should not be in use when using this command to dump from or restore to tape.

If the VOLID of the to-disk is changed by this command, the operator is requested to change the external ID of the disk and the VOLID in the symbolic device entry (SDAT) is cleared. If, as a result of this change, the VOLID duplicates one in use, the proper volume must be used.

The maximum tape record length for 2311 and 3330 dumps is 4096. For a 2314 dump tape, the maximum record length is 8192 bytes. If a dump tape is copied, care must be taken to insure that a full-length record is copied for each record.

When dumping to a tape, if the tape is file protected, the task is abnormally terminated when trying to write a tape label.

DSS? Command

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

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

Note: Manager's and administrators should see Manager's and Administrator's Guide for specialized operands.

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. When two or more data set names are specified, they must be enclosed in parentheses.

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 sharability

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 data set size, record form, and logical record length.

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; and in conversational tasks, the information is printed at the user's keyboard, but the user can terminate the printing at any time by pressing the ATTENTION key.

Programming Notes: The PC? command can be used for a briefer description of the status of cataloged data sets.

When using DSS? nonconversationally, a SECURE command should be issued before DSS? is issued for data sets on private devices.

Examples:

1. The user wants to present the status of his data sets.

```
User:    dss?
System:  NICHOLAS.USERLIB
            SHARED AT LEVEL 02 BY NICHOLAS, ACCESS: RW
         VOLUME: DB0622 (2314)
         ORGANIZATION: VP                    PAGES: 0000071
         REFERENCE DATE: 154/71              CHANGE DATE: 154/71

         NICHOLAS.NICHOLAS.TEST
         VOLUME: DB0668 (2314)
         ORGANIZATION: VI                    PAGES: 0000000
         REFERENCE DATE: 084/71              CHANGE DATE: 084/71
         RECORD FORMAT: V                    RECORD LENGTH: 0000132
         KEY LENGTH: 0000007                 KEY POSITION: 0000004

         NICHOLAS.TA000304.SOURCE.SINGLE
         VOLUME: 014442 (9-TRACK TAPE)
         ORGANIZATION: PS
```

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

```
User:    dss? d.a
```

The system presents the status information.

DUMP Command

This command places the contents and names of specified data fields or expressions in the data set with a data definition name of PCSOUT.

Operation	Operand
DUMP	data field names or expression[,...] id? data field name or expression [,...]

data field name
identifies one or more data fields or expressions to be placed in the PCSOUT data set.

Specified as: the name of a data location, an array, or a control section; a symbolic range, an arithmetic of logical expression, a quoted string, an absolute address, or a command variable.

id? data field name or expression
specifies one or more data fields or expressions for which the CSECT name, load address, and length are to be dumped.

Specified as: the characters id? followed by a data location, an array, an internal or external name, or a symbolic range. (These terms are explained in Section 3 of Part II under "Types of Operand Specification" and "Operand Definitions.") For multiple requests in one DUMP command, the characters ID? must be repeated for each data field or expression. Data fields or expressions preceded by id? and those not preceded by ID? are interchangable in one command.

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 can be only one PCSCUT data set per task. It must be 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
```

PCSOUT must be specified as the data definition name on a DDEF command or a DDEF macro instruction before DUMP is issued. If no definition has been given, the user is prompted to issue one. Refer to Appendix E 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. He enters the following commands:

```
User:  ddef pcsout,vi,dsname=list.pcsout
       dump pgm.csect1
```

The system puts the information in the PCSOUT data set.

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

   The system loads the module and outputs data.

3. The user wants to dump the CSECT name associated with the symbol JOE. He enters:

   User: dump id? joe

   The system displays the CSECT name, load address, and length.

EDIT Command

This command invokes the system's text-editing function for a VISAM or VPAM data set.

Operation	Operand
EDIT	DSNAME=data set name[(member name)][,RNAME=region name] [,REGSIZE=region name length]

DSNAME
identifies the data set to be edited or created.

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.

RNAME
identifies a region, within the data set specified in the DSNAME operand, to be created or edited.

Specified as: an existing region name or a string of from 1 to 244 characters. The value of the REGSIZE operand determines the maximum length of the region name. Region names are padded with blanks or are truncated on the right to fit the specified length.

REGSIZE
specifies the maximum length of each region name for the region data set.

Specified as: a decimal number from 0 to 244. If 0 is specified, a line data set is created.

System default: 0.

Functional Description: The EDIT command invokes the text editor and initializes the transaction table if TRANTAB=Y. Then, EDIT ascertains whether the specified data set or region exists or has previously been defined within the task.

If the data set or region exists, EDIT assumes the existing data definition values and prompts the user with an underscore so that he may enter commands. The CLP is set to the first line of the data set.

If the data set does not exist, EDIT defines it and prompts with the first line number of the new data set (specified by BASE in the user profile; the default is 100) if LINENO=Y. If a region name is specified in the EDIT command, EDIT prompts with the first line of the new region. If the region name is not specified, but REGSIZE is greater than 0, EDIT prompts with an underscore so that the user can specify a region name with the REGION command.

The type of prompt a user gets depends on the value of LINENO, the value of REGISZE, and whether the data set or region is new or old. Table 17 summarizes the prompt.

Table 17. Type of prompt after the EDIT command

LINENO Value	REGSIZE = 0		REGSIZE > 0	
	Data Set is New	Data Set Exists	Region is New	Region Exists
Y	Prompt with the value of BASE	Prompt with an underscore	If a region name is specified, prompt with the value of BASE; otherwise, prompt with an underscore	Prompt with an underscore
N	Open the terminal to accept data	Prompt with an underscore	If a region name is specified, open the terminal to accept data; otherwise, prompt with an underscore	Prompt with an underscore

Programming Notes: The user can precede EDIT with a DDEF command to specify the DCB suboperands he desires. To create a region data set with maximum line length, the DCB values might be:

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

A separate EDIT command must be issued for each data set to be processed. The user can terminate processing of one data set and begin processing another by issuing an EDIT command without an intervening END command.

Following EDIT, the user can issue any command.

Examples:

1. The user wants to create a line data set named LINEDS:

   User: edit lineds
   System: 0000100

   Now, the user can enter data, and the system will continue to prompt with line numbers in increments of 100. To enter a command, for immediate execution, the user must enter the break character followed by the command.

2. The user wants to create a region data set, REGDS, with a region name length of 15 characters. He wants to create region XYZ, and does not want line numbers displayed:

   User: default lineno=n;edit regds,xyz,15

   The system unlocks the keyboard so that the user can enter data at line 100 of region xyz.

   The user can also use the REGION command to operate on other regions in the data set. (See the description of REGION later in this part.)

3. The user wants to edit an existing data set, LINEDS, which begins at line 100. He wants the transaction table active:

   User: default trantab=y;edit lineds
   System: _

   CLP is set to the first line in the data set.

EJECT Command

The EJECT command causes a skip to a new page in the non-conversational SYSOUT listing. When the SYSOUT is a terminal, a triple space is done instead.

Operation	Operand
EJECT	

Note: This command has no operands.

Functional description: The EJECT command module issues a 'GTWRC' macro with an 'EJECT' ASA carriage control character. The EJECT carriage control character is handled differently according to the SYSOUT. For 2741, TTY's and 3215 (1052-7) the paper is spaced up 3 times by issuing three carriage return characters. For SYSOUT datasets with carriage control specified, the carriage control character is included as the first character in the record. When the dataset is printed with the EDIT option, the carriage control character will cause an eject to the next page on the printer. If carriage control is not specified on the SYSOUT dataset, the carriage control character is deleted and a blank line is inserted instead.

Example: The user wishes to start the output from command B on a new page in the SYSOUT dataset.

```
User:     command A
          EJECT
          command B

System:   .....output from A.....
          EJECT to new page
          ......output from B.....
```

ENABLE Command

See "DISABLE, ENABLE, POST, and STET Commands."

END Command

This command terminates processing of a language processor controller (LPC).

Operation	Operand
END	

Note: There are no operands.

Functional Description: When an LPC, such as EDIT, PROCDEF, or PII, is invoked all text editor functions become available. When END is issued the use of these functions is inhibited, and control is passed to the processor's end routine (previously defined when the LPC was invoked). The end processing depends on the particular processor. In the case of EDIT, the data set is closed; whereas, in the case of PROCDEF, the procedure just defined or edited is saved, but the data set is not closed. The function of END for PLI is described under the PLI command.

Programming Notes: The user can terminate processing of one LPC and begin processing another by issuing an EDIT, PROCDEF, or PLI command (preceded by a break character, if necessary) without issuing an intervening END command. The previous LPC's end-processing routine is entered before the new LPC is invoked.

When the system expects data, the END command must be preceded by a break character. To enter an END command preceded by the break character as a line of data, as in a PROCDEF, two break characters must precede the command.

Caution: This command is not the END statement for the Assembler or FORTRAN compiler.

Examples:

1. The user wants to terminate this editing procedure. The data set MYPROG already exists.

```
Sys,User:  edit myprog
Sys,User:  context 700,900,1m,stm
Sys,User:  number 200,last
Sys,User:  end
```

   END, in this case, is not preceded by a break character as the system is expecting a command (it prompted with an underscore).

2. The user creates this data set.

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

   The system expects data for line 300, so END must be preceded by a break character.

3. If the user wants to terminate processing of data set MYDS in example 2 and begin processing a new data set AMYDS, his entry for line 300 is

```
Sys,User:  0000300 _edit amyds
```

ERASE Command

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

Operation	Operand
ERASE	[DSNAME=data set name[(member name)]][,SHARED={Y|N}]

DSNAME
identifies the data set, which resides on direct access storage, to be erased. VAM data sets must be cataloged; and physical sequential data sets 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, or its alias. When specified, the member name or alias is enclosed in parentheses and immediately follows the VPAM data set name.

System default: all data sets qualified by the user's user identification.

SHARED
specifies for shared data sets whether or not an implicit search of the owner's catalog is to be performed (see programming notes).

Specified as:

Y - search.
N - no search.

System default: N.

Functional Description: In conversational mode, when the data set name specified is partially qualified, the ERASE command tests the DEPROMPT operand in the user profile to determine whether each fully qualified data set name referenced by the input name will be presented to the user. If DEPROMPT=Y, the user is presented one fully qualified data set name at a time for disposition. When he responds with E (for erase), the data set name is removed from the catalog, and the direct access storage occupied by that data set is freed. If he responds with R (for retain), no action is taken. If he responds with A (for ALL), all data sets with the specified qualifiers are removed from the catalog. If DEPROMPT=N, all data sets are erased without prompting.

In nonconversational mode, all data sets referenced by the input name are erased.

When the user enters a fully qualified data set name, the data set is erased, regardless of the value of the DEPROMPT operand or the mode of operation.

When the DSNAME operand does not contain a member name, the direct access storage occupied by that data set is freed and the data set name is removed from the catalog.

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

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

When a user, with sharing access U (for unlimited), 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 that requests 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.

Even though the RET command has been invoked to give a user read-only access to a data set, he may still erase that data set.

If DEPROMPT=N and the ERASE command is issued, all data sets are erased without prompting.

Programming Notes: DEPROMPT is initially set to Y. To change its value, the user issues the DEFAULT command, with DEPROMPT as an operand.

The SHARED operand is only meaningful in the situation where ERASE is specified with a partially qualified name and an entry exists in the user's catalog for a data set which was shared at a lower level of qualification. In this case, if SHARED=N is specified, the descriptor entry will be deleted from the sharer's catalog, but the data set entry will remain in the owner's catalog and the data set will not be erased.

Examples:

1. A conversational user wants to erase several data sets whose names begin with A.B. DEPROMPT=Y is in his user profile.

   User: erase dsname=a.b

   The system asks the user to enter E for erase, A for all, or R for retain; it prompts with the fully qualified data set name.

   System: A.B.C
   User: r
   System: A.B.D
   User: r

   The system prompts for next command when all data sets with the fully qualified name have been presented.

2. A conversational user wants to erase all data sets whose names begin with the components E.F.

   User: default deprompt=n
   erase dsname=e.f

The system erases the catalog entries and frees the virtual storage occupied by all data sets with the fully qualified name E.F.

EVV Command

This command catalogs private VAM volumes by volume.

Operation	Operand
EVV	DEVICE={2311|2314|3330|333B}
	,VOLUME=(volume serial number [,...])

Note: Managers and administrators should see Manager's and Administrator's Guide for specialized operands.

DEVICE
specifies the type of direct access device on which the VAM volume resides.

Specified as: 2311 or 2314 or 3330 or 333B (3330-11 disk)

VOLUME
identifies the volume or volumes to be processed.

Specified as: a one- to six-digit volume serial number for each volume. The volume serial numbers are all enclosed in one pair of parentheses.

Functional Description: EVV catalogs only the user's data sets on the volumes specified, but does not open them for use by subsequent programs.

Programming Notes: EVV allows the user to introduce VAM data sets created under other TSS systems into his current installation, or to recatalog his previously deleted VAM data sets. Private VAM data sets created under TSS are automatically cataloged.

Example: The user has three private volumes that are necessary for the execution of his program. He wants to enter them into the system.

User: evv 2311,(111500,111501,111502)

The system makes catalog entries for all of the user's data sets residing on the specified volumes.

EXCERPT Command

This command inserts a region or range of lines from another data set into the current data set.

Operation	Operand
EXCERPT	DSNAME=data set name[(member name)][,RNAME=region name]
	[,N1=starting line[,N2=ending line]]

DSNAME
identifies 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 or must be cataloged. When

DSNAME refers to a VPAM data set, a member name must also be specified.

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.

RNAME

identifies the region from which data is to be inserted into the current data set or region, 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 DSNAME is a line data set. When N1 and N2 are both omitted, the entire data set named in DSNAME 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: a one- to seven-digit absolute decimal number.

System default: If N2 is also omitted, the entire data set -- or region, if RNAME is specified -- is inserted. If N2 is specified, N1 is assumed to be the first line in the specified region or data set.

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-digit decimal number. Can also be specified as LAST to indicate that the lines to be excerpted range from the line specified by N1 to the end of the data set.

System default: the entire data set is inserted when N1 and RNAME are omitted. If RNAME is specified, the entire region is inserted. 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: Insertion is always made immediately after the data line that is the current line location at the time the command is issued. The system automatically renumbers the inserted lines by using the value of INCR. If the existing line numbers do not accommodate the number of lines to be inserted (for example, the user is 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 or region to be excerpted could not be found.
- The line number within the region or data set to be excerpted could not be found.

- The end of the region was reached before any of the requested lines could be excerpted.

Upon completion of this command, the CLP points to the next line following the last inserted line or to the last inserted line plus the value of INCR, whichever is less.

Caution: A language-processing command (EDIT, PROCDEF, or PLI) must be invoked before the command is issued.

Programming Notes: When EXCERPT is preceded by a break character and follows an INSERT command, the excerpted lines are added to the current data set or region. When EXCERPT is preceded by a break character, but follows an EDIT or REGION command, the excerpted lines are added to the data set or region. In this case, the data set or region must be new. If the data set is not new, INSERT must be issued first to position the CLP to a line number other than the first line in the data set or region. When EXCERPT (preceded by a break character) follows REVISE, these lines replace the lines deleted by REVISE.

Examples: The user is editing data set XYZ, which has regions XYZ1 and XYZ2. He wants to excerpt lines from data set ABC (regions ABC1 and ABC2). Lines in all regions are numbered in increments of 100.

1. The user wants to excerpt the entire data set ABC into new region XYZ3.

   User: region xyz3
   Sys,User: 0000100 _excerpt abc

   The system inserts data set ABC into the current region XYZ3.

2. The user wants to excerpt only lines 300 through 500 from region ABC1 in data set ABC to the end of region XYZ1. Assume CLP is positioned in region XYZ1.

   User: insert last
   Sys,User: 0001100 _excerpt abc,abc1,300,500

   The system inserts lines 300 through 500 from region ABC1 at the end of region XYZ1.

3. The user wants to replace lines 200 through 400 in the current region DEF2 with lines 800 through 2000 in region ADD3 of another data set MYDS.

   User: revise 200,400,10
   Sys,User: 0000200_excerpt myds,add3,800,2000

   The system replaces lines 300 through 500 in region DEF2 with lines 800 through 2000 from region ADD3, using an increment of 10.

EXCISE Command

This command deletes a line or a range of lines from the current 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-digit decimal line number that may be absolute or relative.

LAST - last line in the current region.

System default: the value of the CLP.

N2

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

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

LAST - last line in the current region.

System default: only the line specified in N1 is deleted.

Functional Description: After EXCISE is executed, the CLP is set to the value specified in N1. The user is then prompted for a command.

Caution: A language-processing command (EDIT, PROCDEF, or PLI) must be invoked before the command is issued.

Programming Notes: Since the CLP is set to the value of N1, the user may follow this command with either an INSERT or EXCERPT command. The REVISE command is equivalent in function to EXCISE followed by INSERT.

Examples:

1. The user wants to delete line 113 in the current region.

```
User:     excise 113
System:   _
```

2. The user wants to delete the next 10 lines beyond the CLP in the current data set.

```
User:     excise n1=+1,n2=+10
System:   _
```

3. The user wants to delete all lines between 100 and 300 in the current region.

```
User:     excise 101,299
System    _
```

EXECUTE Command

This command introduces a nonconversational task into the system.

Operation	Operand
EXECUTE	DSNAME=data set name

DSNAME

identifies the VSAM (fixed-format or variable-format) data set or the VISAM line data set that resides on public storage and that contains a series of commands starting with LOGON and ending 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 BSN is assigned for the task, 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. Each SYSIN data set represents one task.

The EXECUTE command differs from the BACK command in these ways:

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 and must reside on public storage. The data set specified in the BACK command need only conclude with a LOGOFF command and can be private or public.

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 canceled, 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 created in a data set named NEWTASK.

```
User:       edit newtask
Sys,User:   0000100 logon user01
            0000200 asm progx,y,isd=y
            0000300 logoff
            0000400 _end
            execute newtask
```

The system accepts the task and assigns a BSN.

EXHIBIT Command

This command allows the user to determine the status of any batch or bulk I/O job he has initiated or to obtain a list of all currently active system users.

Operation	Operand
EXHIBIT	OPTION1={BWQ[,TYPE={ALL|BSN.number}]| UID[,TYPE={CONV|BACK|UID.userid|ALL}]}

Note: For special operands, see Manager's and Administrator's Guide or Operator's Guide.

OPTION1
specifies whether to display batch work queue activity, or active user task status.

Specified as:

BWQ - batch work queue status.
UID - active user task status.

TYPE
specifies the data to be displayed.

Specified for BWQ as:

ALL - All BWQ entries, for the user, are displayed.

BSN.number - The entry assigned to the BSN is displayed. (The BSN is a decimal number from 257 to 9999.)

System default: ALL.

Specified for UID as:

CONV - All conversational tasks are displayed.

BACK - All nonconversational tasks are displayed.

UID.userid - All tasks are displayed for the specified USERID (from three to eight alphameric characters).

ALL - All active tasks are displayed.

System default: ALL.

Functional Description: EXHIBIT displays BWQ or user information. For OPTION1=UID, the following information is displayed:

USERID -- The user's identification.
TID -- The task number assigned to the task.
TYPE -- Either CONV for conversational or BACK for nonconversational.
SYSI -- Either a symbolic device address of SYSIN/SYSOUT for a conversational task or a BSN for a nonconversational task.

For OPTION1=BWQ, the following information is displayed.

BSN -- Batch sequence number.
USERID -- The user's identification.
TID -- The task number assigned to the task.
TYPE -- Batch request type: LIST (print), EXECUTE, PUNCH, RTAPE, or WTAPE.
STAT -- Status of the job request:

A -- active
P -- pending (awaiting execution)
C -- canceled
S -- shutdown
F -- Erase

DEV -- Type of device required for the job (for example, U/R unit record).
STAID -- Station identification for Remote Job Entry (RJE) jobs.
DSNAME -- Data set named for the job (up to 35 characters).

Caution: After the user issues the EXHIBIT command, his catalog contains a pointer to the SYSUBWQ data set. This data set contains the BWQ information displayed by EXHIBIT. The user may delete this pointer by issuing:

```
delete sysubwq
```

Example:

1. The user wants to display all active tasks.

```
User:    exhibit option1=uid,type=all
System:  ACTIVE USER STATUS AT 10:31:59    06/11/7X
         USERID   TID  TYPE SYSI

         USERID01 0056 CONV 0089 USERID02 0057 CCNV 0090
         USERID06 0060 BACK 0259
```

2. The user wants to display the status of his nonconversational job that has BSN=262.

```
User:    exhibit bwq,type=bsn.262
System:  BATCH WORK QUEUE STATUS AT 10:34    06/11/7X
         BSN  USERID    TID  TYPE  STAT  DEV  STAID     DSNAME
         0262 USERID03  057  LIST  P     U/R  RJESTA01  DSNAME.HIS
```

EXIT Command

This command bypasses execution of the current program or command, and the next command in the source list is executed.

Operation	Operand
EXIT	[SIRTEST={Y|N}]

SIRTEST
specifies whether the system checks for a user-defined SIR routine.

Specified as:

N - system does not check.

Y - system checks for a user-defined SIR routine. If one exists, the EXIT function is canceled.

System default: N.

Functional Description: The EXIT command returns control to the next command or program in the current source list. Any command that follows EXIT is ignored. After the current source list is processed, the system prompts for a command. If the user then enters the GO command, a previously interrupted source list is executed.

If the SIRTEST parameter is set to Y, the system first checks for any active user-defined SIR routines. If one is active, the EXIT command is canceled. When the CLEANUP implicit operand is set to Y, EXIT cancels all user-defined AETD and SIR routines. An exit from an AETD routine causes control to return to the program that was processing when the AETD routine was invoked by an attention interruption. (AETD and SIR routines are explained in Assembler User Macro Instructions.)

Example: The user interrupts a command string containing three program calls (PROGA, PROGB, PROGC); the interruption occurred in PROGB. He then issues the EXIT command:

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

The system terminates current command processing -- PROGB -- and passes control to the next program in the source list --PROGC.

If the user had interrupted a source list before he entered the source list containing calls to PROGA, PROGB, and PROGC, he could resume processing at that earlier source list by issuing the GO command after the system had processed PROGA, PROGB, and PROGC.

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	{MSGID | ORIGIN | word | TEXT | RESPONSE | [,message identification] | MSGE | MSGS}

MSGID
: the identification of some message in the message file (SYSMIF). Indicates that the user wishes to see the message identification. Only the message ID will be shown.

  Specified as: MSGID

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 system programmers.

word
: specifies that a word, within the text of the last message, is to be explained.

  Specified as: a normal or quoted string.

TEXT
: specifies that a code-identified message will be displayed in full text.

  Specified as: TEXT

RESPONSE
: specifies that the possible responses to the last message are to be displayed.

  Specified as: RESPONSE

MSGE
: indicates that the user wants the extended message to be displayed.

  Specified as: MSGE

MSGS
: indicates that the user wants the standard message displayed.

Specified as: MSGS

message identification
specifies a message ID for which a word, text, response, extended message, or standard message is to be displayed. This form is used when the request is for a message other than the most recently issued message.

Specified as: a one- to eight-character message ID.

Functional Description: If no operand is used with the EXPLAIN command, the preceding message is restated more explicitly. If the message is not explainable but contains explainable words, these words are elaborated; otherwise, the system's reply is "no explanation."

Programming Notes: Only one of the options may be specified when an EXPLAIN command is issued. If the user wants to use more than one option, he must give additional EXPLAIN commands.

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. CRG.
```

The user does not understand what is required, so he requests an explanation:

```
explain
```

The explanation for the current message UPTCM170 is displayed:

```
UPTCM170 A VAM DATA SET'S ORGANIZATION DEFINES THE OVERALL RELATICN-
SHIPS 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:

```
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 with the message ID is necessary here because the explanation requested is not for the most recently issued message.

FILEDEF Command

This command defines a dataset and describes its characteristics to the system (DDEF). Additionally, it provides the link between TSS and OS ddnames for the Program Product Language Interface.

Note: The operands with the exception of MACRO, OSDDN and OSKEYLE are identical to DDEF operands.

Operation	Operand
FILEDEF	DDNAME=ddname,DSORG=VI|VS|VP[,DSNAME=dsname...] [,MACRO=CONC] [,OSDDN=osddname] [,OSKEYLE=number]

DDNAME,DSORG,DSNAME,...
these are identical to DDEF operands defined in this manual.

MACRO
signifies that the dataset specified in the DSNAME operand is to be concatenated with another dataset with the same OS ddname (OSDDN parameter). The OSDDN parameter is required in this case because the TSS ddname (DDNAME option) cannot be the same for both datasets. See example.

Specified as: CONC

OSDDN
specifies the OS ddname with which the TSS dataset is to be associated. If no OSDDN is specified, it is assumed to be the same as the TSS ddname. If the ddname is specified as SYSxxx, the TSS ddname will not be changed to TSSxxx, but the OS ddname will remain SYSxxx.

OSKEYLE
specifies the length of the key. This parameter is required when OS/VS BDAM or ISAM access methods are being simulated.

Functional description: The FILEDEF command causes a DDEF to be issued against the TSS dataset specified with the ddname specified. It also causes a control block to be built which will be the link between the OS dataset specification and the TSS specification. The DDEF is issued exactly as written in the first operands of the FILEDEF parameter list.

Examples: FILEDEF a,vp,dset1

A control block is created relating the TSS dataset ("dset1") and a TSS ddname of "a" with an OS description of that dataset with an OS ddname also of "a".

```
FILEDEF b,vi,dset2,OSDDN=dcbout
```

In this case the TSS and OS ddnames are different but the control block built associates them at execution time.

```
FILEDEF mac1,vp,macbb1,OSDDN=syslib FILEDEF

mac2,vp,macbb2,OSDDN=syslib,macro=conc FILEDEF

mac3,vi,macbb3,OSDDN=syslib,macro=conc
```

This example shows how to "concatenate" three macro libraries to be read by one OS read statement and one OS ddname. The last one filedeffed

will be searched for the requested member and if it is not found, the previous will be searched, etc. The TSS libraries may be partitioned or region datasets. This facility is only valid if the OS partitioned access method is used by the executing program.

A FILEDEF command referring to the same dataset name as a previous FILEDEF will replace the old ddname with the new ddname.

FILEREL Command

This command deletes the data definition established by a previously issued FILEDEF command. It also disconnects the OS/TSS link.

Operation	Operand
FILEREL	OSDDN=osddname

OSDDN
identifies the dataset definition created by a FILEDEF command.

Functional description: FILEREL will cause a RELEASE to be done on each TSS ddname associated with the specified OS ddname. It also causes the control block linking the os dataset specification with the TSS specification to be destroyed. This will not be done unless all DCB's against the datasets are closed.

Example: FILEREL syslib

This will cause all filedef's associated with the OS dataset specification to be released as well as destroying the linking control blocks.

FTN Command

This command invokes the FORTRAN compiler to compile a source program module.

Operation	Operand
FTN	NAME=module name[,STORED={Y|N}][,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}] [,LISTDS={Y|N}][,LINCR=(first line number,increment)]

NAME
identifies the object module to be created. If the source program module (that is, source language data set) is prestored, the user must have named it:

SOURCE.name

If it is not prestored, the system automatically prefixes SOURCE. to the source program module name. The listing data set is automatically 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 not be identical to other external entry points in the library in which it is stored. See FORTRAN Programmer's Guide for a complete list of naming rules.

STORED
specifies whether the source program module is prestored.

Specified as:

Y - source program is prestored.
N - source program is not prestored.

System default: N.

VERID
specifies the version identification to be assigned to the object program module.

Specified as: from one to eight alphameric characters.

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.

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.

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.

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.

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.

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.

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.

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.

LISTDS
determines whether the user-requested listings from the language processors are to be placed in a list data set or placed directly on SYSOUT.

Specified as:

Y - placed in list data set.
N - listings to SYSOUT.

System default: Y.

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-digit decimal numbers, separated by a comma and enclosed in parentheses; the last two digits in each number must be zeros.

System default: (100,100).

Note: This operand is ignored when STORED=Y.

Functional Description: See "Language Processing" in Section 3 of Part II.

Caution: The command is canceled if invalid operands are entered.

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. The following program multiplies two digits:

```
Sys,User:  ftn raader,slist=y,oblist=y,crlist=y,isd=n
           0000100     read (5,10) a,b
           000020010   format (1x,2f6.2)
           000030020   format (3f10.3)
           0000400     atb=a*b
           0000500     write (6,20) a,b,atb
           0000600     stop
           0000700     end
System:    _
```

FTNH Command

This command will invoke the FORTRAN H EXTENDED program product using the Program Product Language Interface.

Operation	Operand
FTNH	NAME=modulename [,CSOPTS=(opt1,opt2,...)] [,SOURCEDS=sourcedsname]

NAME
identifies the name by which the object program will be known to TSS. It consists of one to eight alphameric characters, the first of which is alphabetic. If the SOURCEDS option is not specified, there must exist a dataset called SOURCE.name which is assumed to be the source program to be compiled.

OSOPTS
specifies a list of OS options to be in effect during the compilation.

Form	Abbreviated Form	Form	Abbreviated Form
SOURCE|NOSOURCE	S|NOS	XREF|NOXREF	

LINECOUNT(number)[1]	LC(number)	NAME(name)[6]	
LIST|NOLIST		EBCDIC|BCD	EB|BCD
OBJECT|NOOBJECT[2]	OBJ|NCOBJ	SIZE(MAXnnnK)	
DECK|NODECK			
OPTIMIZE(0|1|2)[3]	OPT(0|1|2)		
NOOPTIMIZE	NCOPT	AUTODEL(value)	AD(value)
FORMAT|NOFORMAT[4]	FMT|NOFMT	ALC|NOALC	
GOSTMT|NOGOSTMT[5]		ANSF|NOANSF	
MAP|NOMAP		FLAG(I)|FLAG(E)|FLAG(S)	

[1] Compiler also accepts the old form: LINECNT=xx

[2] Compiler also accepts the old form: LOAD|NOLOAD

[3] Compiler also accepts the old form: OPT=0|1|2

[4] Compiler also accepts the old form: EDIT|NOEDIT

[5] Compiler also accepts the old form: ID|NOID

[6] Compiler also accepts the old form: NAME=name

Additional information is available in Appendix K and the CS FORTRAN H EXTENDED Programmer's Guide.

SOURCEDS
specifies the name of the input dataset (SYSIN to CS) if source. module is not to be used.

GAV Command

This command searches the entire Combined Dictionary for whatever types of entries the user has specified and presents the data on the user's SYSOUT.

Operation	Operand
GAV	[TYPE={SYN|DEF|CSW}]

TYPE
identifies the type of search.

Specified as: SYN (Synonyms), DEF (Defaults), or CSW (Command Symbol Words).

System Default: All three types will be processed.

Functional Description: The Combined Dictionay will be searched for the type of entry specified. If TYPE is defaulted, a search will be performed for all three types. Output will be presented at the user's SYSOUT in the form 'TERM VALUE' for synonyms and defaults; for command symbol words, only the names will be listed, though the kind of CSW will be indicated by the header.

Examples: If SYNONYM B = BARB and SET A = 8 had been previously entered during the task, and no other synonyms or command symbol words exist for this task, then;

```
User:    gav  syn
System:  ***SYNONYMS***
          B       BARB

User:    gav  csw
System:  ***INTEGER CSWORDS***
         A
```

GDV Command

This command will list on the user's SYSOUT the default value associated with a specified term.

Operation	Operand
GDV	DFLT = term

DFLT
is the term which is to be searched for.

Specified as: 1-8 character name.

Default: None

Functional Description: The user's Combined Dictionary will be searched for the specified term. If this term is found, its current default value will be listed. If not, the message "THERE IS NO DEFAULT" will be issued.

Example;

```
User:    default base = 150
User:    gdv  base
System:  150
```

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.

The GO command issues a message stating the location at which execution continues.

Caution: In a dynamic statement, GO is meaningless; a diagnostic message is issued. In an immediate statement, GO must appear last, because the commands following are ignored; no diagnostic message is issued.

Programming Notes: GO is meaningful when it follows an attention interruption or after PCS has been used to STOP execution. Otherwise, the command is canceled.

Example: In executing his program ABC, the user wants to interrupt execution and modify his program.

```
User:     call abc
```

The system invokes ABC.

```
User:     (presses ATTENTION key)
System:   !
User:     set 5r=6;go
```

The system resumes executing ABC from the point of interruption.

GOTO Command

The GOTO command provides the ability to branch forward in PROCDEFs.

Operation	Operand
GOTO	{command|OUT|'comment'}

command|'comment'
specifies the command statement to which control is to be passed.

Specified as: a command name or comment no longer than 8 characters beginning immediately after the underscore or semicolon of the destination statement.

OUT
specifies that an immediate return to the calling procdef is to be made.

Functional Description: The GOTO command is to be used to branch forward in a PROCDEF or nest of PROCDEF's. The GOTO command routine searches the source list until 1) the destination is found or 2) the end of the PROCDEF is encountered or 3) the end of the nest of PROCDEF's is encountered. In case 2, if the destination was OUT the search stops and return is made as if the PROCDEF had completed normally. If the destination was not OUT, the search continues until case 1 or 3 occurs. If 3) occurs, control is returned to the command system.

Programming Notes: The destination operand is to be specified as command name or as 'comment'. The command or comment must begin in the first column of the PROCDEF line or immediately after a semicolon if multiple commands per line are used. The destination operand may also be specified as input to the procdef - i.e., GOTO $command or GOTO '$comment'. In addition, if GOTO OUT is specified, control will be returned to the program calling the PROCDEF (the command system or another PROCDEF) as if the PROCDEF had completed normally. If the command or comment cannot be found within the PROCDEF in which the GOTO exists (except for OUT as above), control will be returned to the command system. Note that the command or comment must be local to the GOTO or in a higher level PROCDEF, i.e., you cannot "GOTO" a command or comment in a lower level PROCDEF. The GOTO command may be used as the object command in an IF statement.

Examples:

The user enters the following PROCDEFs:

```
User:       procdef sample1
Sys,User:   0000100 param $1
            0000200 display 'begin sample1'
            0000300 if '$1'='3';goto 'call'
            0000400 if '$1'='1';goto 'label'
            0000500 if '$1'='2';goto display
            0000600 goto out
            0000700 'label' display 'one';goto 'call';display 'two'
            0000800 'call' sample2 $1
            0000900 display 'return from sample2'
            0001000 _end

User:       procdef sample2
Sys,User:   0000100 param $1
            0000200 if '$1'='3';goto 'label'
            0000300 if '$1'='2';goto 'two'
            0000400 display 'error in input for sample2'
            0000500 goto xxx
            0000600 'label' display 'sample2-three'
            0000700 goto out
            0000800 'two' display 'sample2-two'
            0000900 _end
```

If the PROCDEFs specified above were issued with the following inputs, the outputs would be:

1.
```
User        sample1 1
System:     BEGIN SAMPLE1
            ONE
            ERROR IN INPUT FOR SAMPLE2
            -
```

The GOTO xxx statement in sample2 causes an exit from all PROCDEFs, not a return to PROCDEF sample1.

2.
```
User:       sample1 2
System:     BEGIN SAMPLE1
            TWO
            SAMPLE2-TWO
            RETURN FROM SAMPLE2
            -
```

The GOTO display in sample1 will not cause a branch to 'LABEL' display 'one' because "display" does not begin in the 1st column of the command line.

3.
```
User:       sample1 3
System:     BEGIN SAMPLE1
            SAMPLE2-THREE
            RETURN FROM SAMPLE2
            -
```

GOTO 'label' in sample2 does not cause a branch to 'label' in sample1. Also GOTO out in sample2 causes a return to the sample1 PROCDEF.

```
User:       sample1 x
System:     BEGIN SAMPLE1
            -
```

GOTO OUT in sample1 causes an exit from the PROCDEF.

GSV Command

This command will list the synonym value associated with a specified term, or all terms associated with a specified synonym value.

Operation	Operand
GSV	NAME = value or term [,SEARCH={T|V}]

NAME
identifies the value or term which is to be searched for in the user's Combined Dictionary.

Specified as: if a term is specified - a 1-8 character name. If a value is specified, a 1-244 character name.

SEARCH
identifies the type of search that is to be performed.

Specified as: T or V

System default; V

Functional Description: If SEARCH=T, the Combined Dictionary will be searched for the specified term. If the term is found, the value for the term will be listed on the user's SYSCUT. If SEARCH=V, or if SEARCH is defaulted, the entire Combined Dictionary will be searched for all terms associated with the specified value. The term(s) and value will be presented on the user's SYSOUT.

Programming notes: To avoid confusion between 'term' and 'value' -- remember that synonyms are entered in this manner:

```
SYNONYM  term = value.
```

Examples: If SYNONYM L = LIST had been entered previously, then:

```
User:    gsv  l, t
System:  LIST

                    OR

User:    gsv  list
System:  L       list
```

HASM Command

This command causes the OS ASM H Program Product to be invoked using the Program Product Language Interface.

Operation	Operand
HASM	NAME=module name[,OSOPTS=(opt1,opt2,...)] [,SOURCEDS=sourcedsname]

NAME
identifies the name by which the object program will be known to TSS. It consists of one to eight alphameric characters, the first of which is alphabetic. If the SOURCEDS option is not specified,

there must exist a dataset called SOURCE.name which is assumed to be the source program to be compiled.

OSOPTS
specifies a list of OS options to be in effect during the compilation.

Each of these options has a standard or default value which is used for the assembly if you do not specify an alternative value.

The option list must not be longer than 100 characters, including the separating commas. You may specify the options in any order. If contradictory options are used (for example, LIST and NOLIST), the rightmost option (in this case, NOLIST) is used.

The assembler options are:

[DECK|NODECK] [,OBJECT|NOOBJECT] [,LIST|NOLIST]

[,TEST|NOTEST] [,'XREF(FULL|SHORT)'|NOXREF]

[,'LINECOUNT(nn)'] [,ALIGN|NOALIGN] [,RENT|NORENT]

[,ESD|NOESD] [,RLD|NORLD] [,BATCH|NOBATCH]

[,'SYSPARM(string),FLAG(nnn)']

DECK
text cards are written to the LOAD.module dataset in preparation for object deck conversion.

OBJECT
text cards are placed in the dataset specified as PUNCH.module.

Note: The OBJECT and DECK options are independent of each other. Both or neither can be specified, but only the output of DECK is used for conversion to a TSS formatted module.

ESD
the assembler produces the External Symbol Dictionary as part of the listing.

RLD
the assembler produces the Relocation Dictionary as part of the listing.

BATCH
the assembler will do multiple assemblies under the control of a single set of job control language cards.

LIST
an assembler listing is produced. Note that the NCLIST option overrides the ESD, RLD, and XREF options.

TEST
the object module contains the special source symbol table required by the test translator (TESTRAN) routine.

XREF (FULL)
the assembler listing will contain a cross reference table of all symbols used in the assembly. This includes symbols that are defined but never referenced. The assembler listing will also contain a cross reference table of literals used in the assembly.

XREF (SHORT)
the assembler listing will contain a cross reference table of all symbols that are referenced in the assembly. Any symbols defined but not referenced are not included in the table. The assembler listing will also contain a cross reference table of literals used in the assembly.

RENT
the assembler checks for a possible coding violation of program reenterability.

LINECOUNT (nn)
the number of lines to be printed between headings in the listing is nn. The permissible range is 1 to 99 lines.

NOALIGN
the assembler suppresses the diagnostic message "IEV033 ALIGNMENT ERROR" if fixed point, floating-point, or logical data referenced by an instruction operand is not aligned on the proper boundary. The message will be produced, however, for references to instructions that are not aligned on the proper (halfword) boundary or for data boundary violations for privileged instructions such as IPSW. DC, DS, DXD, or CXD constants, usually causing alignment, are not aligned.

ALIGN
the assembler does not suppress the alignment error diagnostic message; all alignment errors are diagnosed.

FLAG (nnn)
error diagnostic messages below severity code nnn will not appear in the listing.Diagnostic messages can have severity codes of 0, 4, 8, 12, 16, or 20 (0 is the least severe). MNOTEs can have a severity code of 0 through 255.

For example, FLAG (8) will suppress messages for severity codes 0 through 7.

SYSPARM (string)
'string' is the value of the system variable symbol &SYSPARM. The assembler uses &SYSPARM as a read-only SETC variable. If no value is specified for the SYSPARM option, &SYSPARM will be a null (empty) character string. The function of &SYSPARM is explained inthe Assembler H Language Specifications and in OS/VS and DOS/VS Assembler Language.

You cannot specify a SYSPARM value longer than 56 characters. Two quotes are needed to represent a single quote, and two ampersands to represent a single ampersand.

SYSPARM ((&&AM,''EO).FY)
assigns the following value to &SYSPARM:

(&AM,'EO).FY.

Any parentheses inside the string must be paired.

Note: Even though the formats of some of the options previously supported by Assembler H have been changed, you can use the old formats for the following options: ALGN (now ALIGN), NOALGN (NOALIGN), LINECNT=nn (LINECOUNT(nn)), LOAD (OBJECT), NOLOAD (NOOBJECT). MULT (BATCH), NOMULT (NOBATCH), XREF (XREF(FULL)), MSGLEVEL=nnn (FLAG(nnn)).

Default Options: If you do not code an option, the assembler assumes a default option. The following default options are included when Assembler H is shipped by IBM:

```
(DECK,NOOBJECT,LIST,NOTEST,'XREF(FULL),LINECOUNT(55)',ALIGN,NOBATCH
,'SYSPARM(),FLAG(0)')
```

However, these may not be the default options in effect in your installation. The defaults can be respecified when Assembler H is installed. For example, NODECK can be made the default in place of DECK. Also, a default option can be specified during installation so that you cannot override it.

For further information, please refer to the OS Assembler H Programmer's Guide.

SOURCEDS
specifies the name of the input dataset if SOURCE.module is not used.

Functional Description: HASM will invoke the ASM H program product. The interface routine invoked by the HASM command will filedef all required datasets. PPLI will allow processing of a TSS region dataset in place of a VP dataset (PO in OS/VS terms). The filedefs may be overriden by issuing the appropriate filedef commands before entering HASM.

One of the outputs of the HASM process will be a LOAD.name dataset. This dataset will be input to the object deck converter (CESHR), which will convert it to a TSS loadable module. Additional output is a LIST.name dataset which may be printed at the user's discretion.

IF Command

This command, included in a command statement, specifies 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 designate any valid condition.

Operation	Operand
IF	condition

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 is a dynamic statement, 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 one when the specified instruction location is reached, whether or not the IF condition is true. (See "Types of Operand Specification" in Section 3 of Part II.) When more than one IF command appears in the same conditional statement, the commands in the statement are executed from left to right until an IF that specifies an unsatisfied condition is encountered, or the end of the statement is reached. For example:

```
if X<0;display X;if Y<0;display Y
```

X will be displayed whenever X is less than 0, but Y will be displayed only 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 is 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 may be used to control the frequency at which, or the interval through which, the statement containing IF is effective. In statements other than dynamic 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

   The system evaluates the logical expression and executes DISPLAY only if the condition is true.

2. The user wants to execute more PCS commands every fifth time.

   User: at p.x;if % = (%/5)*5;

   The statement is not evaluated at this time. The system assigns a number to the dynamic statement.

INSERT Command

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

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

N1

identifies the line that is to be the first line or that is to precede data lines that are to be inserted in the current region or data set.

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

LAST - last line in the current region.

System default: the value of the CLP.

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: from one to seven decimal digits. An all-zero increment is not allowed.

System default: 100.

Functional Description: If N1 (the CLP, if N1 is defaulted) does not exist, the first line is inserted at N1. The increment is then added for all subsequent lines inserted. If N1 exists, the first line is inserted at N1 plus the value of INCR. If this line also exists, the command is canceled. When inserting lines between two existing lines, the insertions are made until a new line would overlay an existing line or until the new line is greater than the limiting line. When either of these situations occurs, the command is canceled.

INSERT prompts the user with line numbers for his insertions. Each time the RETURN key is pressed, a line number incremented by the value of INCR is issued. To terminate INSERT processing, enter a command preceded by a break character. The CLP is set to the last line entered plus the value of INCR. If adding INCR to the last line entered exceeds the next existing line, CLP is set to the next existing line. If no data lines are entered, the CLP is set to N1.

Caution: INSERT does not overlay an existing line with a new line. A language-processing command (EDIT, PROCDEF, or PLI) must be issued before the command is issued.

Programming Notes: INSERT is provided for consecutive insertions. UPDATE should be used for the insertion of arbitrary line numbers.

Examples:

1. The user wants to insert data lines, in increments of 10, following line 600.

   User: insert 600,10
   System: 0000610

   Note: Assuming that line 700 is the next existing line number after line 600, nine lines can be inserted.

2. The user wants to insert lines in the data set 10 lines beyond the CLP, with an increment of 100. The CLP is line 500.

   User: insert n1=+10
   System: 0001500

3. The user wants to add lines to the end of an existing data set.

   User: insert last

   The system prompts with the last line number plus the value of INCR.

4. The user wants to generate a region with a blank name in a new region data set.

   User: default regsize=8
   edit myds
   Sys,User: insert
   System: 0000100

   Note: The REGION command must be issued to generate a region with a blank name in an existing region data set. EDIT automatically positions the CLP to the first region in an existing data set, and INSERT, without operands, assumes this value for the CLP.

JOBLIBS Command

This command gives the user the ability to move any one of his JOBLIBs to the logical top of the JCBLIB chain.

Operation	Operand
JOBLIBS	DDNAME=data definition name

DDNAME
: specifies the DDNAME of the DDEF used to define the JOBLIB to be moved to the top of the JOBLIB chain. If defaulted, a diagnostic is issued and the command is canceled.

Functional Description: Used to move the specified JOBLIB and its associated DCB from its present position in the chain to the logical top of the chain. This positioning is important when the loader and compilers retrieve or store modules.

Example: The user wants to store the next object program he assembles in the data set LIB1, which he defined as a JOBLIB earlier in his terminal session. LIB1 has a DDNAME of A1 and currently is not the top JCBLIB in the chain. The user enters:

User: joblibs a1

The system moves the data set LIB1 to the top of the JOBLIB chain.

K, KA, and KB Commands

These commands transfer control from the 1056 Card Reader to the attached 1052 Printer-Keyboard. The KA and KB commands also control the user's input character set.

Operation	Operand
K	

Operation	Operand
KA	

Operation	Operand
KB	

Note: These commands have no operands.

Functional Description: The K, KA, and KB commands indicate to the system that input will come from the user's 1052 Printer-Keyboard. To use these commands, the user can include a K, KA, or KB card in his card deck in the 1056 Card Reader; when the system reads this card, control will return to the attached 1052 Printer-Keyboard. The user can also use the KA and KB commands to change the input character set while he is entering commands from a terminal. The commands function as follows:

K -- transfer control from the 1056 Card Reader to the attached 1052 Printer-Keyboard. If the card reader mode was CA, the terminal mode is KA; if the card reader mode was CB, the terminal mode is KB.

KA -- transfer control to the printer-keyboard and use the full EBCDIC character set (that is, uppercase and lowercase characters are used as such; no folding takes place).

KB -- transfer control to the printer-keyboard and use the folded EBCDIC character set (see below).

The input character set is determined by the value of the ALPHABET operand: when the user first logs on to the system or enters the KB command, the ALPHABET=1, which indicates that the folded character set is used for input. When the user enters the KA command, ALPHABET=2, and the full EBCDIC character set is used. In the folded mode, lowercase letters are converted to their uppercase equivalents. In the full or "unfolded" mode, lowercase letters are not converted; they are used as they are entered. In either mode, the special characters, , ", !, @, #, and $ are valid alphabetic characters. They are never folded. Note that system-supplied commands are coded in uppercase letters. If the user is in KA mode, he must shift to upper-case to execute these commands.

The values set with the KA and KB commands are in effect only for the duration of the user's task; however, if a PROFILE command is entered, following a KA or KB in the same task, the values are in effect for subsequent tasks until the value is changed. For example, if the user is in KB mode and enters the KA command, he will be in KA mode only for the duration of the task. When he logs on again he will be back in KB mode. If, however, he entered a PROFILE command later in the same task, he will be in KA mode when he logs on for subsequent tasks. (See the description of the PROFILE command in Section 6 of Part II.)

Examples:

1. The user wants to have the system take input from the 1052 Printer-Keyboard when the last card is read from the 1056 Card Reader (see C, CA, and CB Commands). To do this, the user includes a K, KA, or KB card as the last card in his input deck. When the system reads this card, it goes to the 1052 Printer-Keyboard for input.

2. The user wants to change his input character set from folded mode to unfolded mode. He is already entering input from the terminal.

   User: KA

   The system accepts input in full EBCDIC mode.

KEYWORD Command

This command displays command names, and their operands, from the user's command library -- USERLIB(SYSPRO) or from the system's command library -- SYSLIB(SYSPRO).

Operation	Operand
KEYWORD	[PROCNAME=command name]

PROCNAME
specifies a command name for which the user wants the operands displayed.

Specified as: a valid command name.

System default: all commands and operands are displayed from the user's command library -- USERLIB(SYSPRO).

Functional Description: The KEYWORD command displays command names and their associated operands from the user's command library. It displays the command name, all the parameters included on the PARAM line of a PROCDEF, or all the keywords defined in the BPKD macro instruction for a BUILTIN-defined command. If the user does not include a command name as the operand of the KEYWORD command, all the command names and their operands are displayed from the SYSPRO member of the user's USERLIB data set. If the user specifies a command name that does not exist, a diagnostic is issued and the command is canceled.

Programming Notes: The KEYWORD command also displays command names and operands from the system command library -- SYSLIB(SYSPRO). If you enter a command name as the operand of the KEYWORD command, the system first searches USERLIB for the command. If the command is not there, the system searches its own command library (in SYSLIB). However, some system commands are not displayed. You are not able to use some of the displayed parameters because either they are not permitted to your privilege class or they are dummy parameters used for system processing only. Also, some commands (SYNONYM, DEFAULT, and the PCS commands) do not use the system's parameter processing facilities; the parameters for these commands are seen only by the associated command processing routines. When you enter one of these commands as the operand of the KEYWORD command, the system does not display any parameters. (In these cases, consult this manual for operand specifications.)

Examples:

1. The user wants to display all commands and operands from his command library:

   User: keyword

   The system displays the command names and operands from the user's USERLIB(SYSPRO data set.)

2. The user wants to see the operands for the FRAMIS command:

   User: keyword framis
   System: FRAMIS, PARAM1, PARAM2, PARAM3

LINE? Command

This command presents one or more lines from a line data set to SYSOUT.

Operation	Operand
LINE?	DSNAME=data set name[(member name)] [,{line number|(first line number,last line number)}[,...]]

DSNAME
identifies a line data set that must be defined by a DDEF command within the current task or must be cataloged.

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, immediately following the data set name.

line number
identifies a single line to be displayed from the specified data set.

Specified as: a one- to seven-digit decimal number.

System default: if the "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-digit decimal numbers, separated by a comma and enclosed in parentheses.

System default: if the "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 first line number that does not exist but is within the bounds of the data set, the next-higher line is presented.

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.

The format of output for line data set is as follows:

Position	Contents
1-7	line number
8	blank if line was created from terminal keyboard; C if line was created from card reader
9	text

The format of output for language processor listing data set is as follows:

Position	Contents
1-130	text (record positions 2 through 131)

Caution: In the specification of a range of line numbers, the first line number must be less than or equal to the last line number. A maximum of 10 line-number ranges may be specified in a single execution of the LINE? command.

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

The system presents the specified lines.

2. The user wants lines 900 through 2400 and lines 4400 through 16000 of member AB1 of data set REPLAY to be presented.

   User: line? replay(ab1),(900,2400),(4400,16000)

   The system presents the specified lines.

3. The user wants his entire data set, LIST.PLAYER, to be presented.

   User: line? list.player

   The system presents contents of the data set.

LIST Command

This command displays a line, or range of lines, or the value of the CLP at the user's terminal or SYSOUT.

Operation	Operand
LIST	[N1={starting position|CLP}][,N2=ending position] [,CHAR={C|H|M}]

N1

identifies the line, or first of a range of lines, or the value of the CLP in the current region to be displayed.

Specified as: a one- to seven-digit decimal 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 1) position of data 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, including record keys.

N2

identifies the last line in a range of lines of the current region to be displayed.

Specified as: a one- to seven-digit decimal 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. This ending character is included in the display.

System default: when N1 is specified, it is the only line listed. Otherwise, the entire data set is listed.

CHAR
specifies type of output.

Specified as:

C - character.
H - hex.
M - mixed.

System default: C.

Functional Description: LIST displays entire lines or the specified portions of lines. If LINENO=Y, region names and line numbers are included. If character positions other than the first or last positions are specified for N1 or N2, these positions apply to all lines in the specified range. 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 plus the value of INCR. The user is then prompted for a command.

The mode of the output data is determined by the CHAR operand in the LIST command:

CHAR=C - all printable characters are displayed in character notation; unprintable characters are ignored.

CHAR=H - the line is displayed in hexadecimal notation.

CHAR=M - all printable characters are displayed in character notation; unprintable characters are displayed in hexadecimal notation, and are underlined.

Caution: A language-processing command (EDIT, PROCDEF, or PLI) must be invoked before the command is issued.

If CHAR=M, and if the user has altered his output translation table, the results of the display are unpredictable.

Examples: The user has previously issued a REGION command to create the following data set:

```
User:      region anyregn
Sys,User: 0000100 line 1
          0000200 line 2
          0000300 line 3
```

1. Assuming the CLP is 400, the user can issue any of these LIST commands to display the entire data set.

```
User:      list 100,300
             or
           list -3,last
             or
           list -5,+1
```

(Note: When N1 and N2 exceed the limits of the data set, the lowest and highest line numbers in the data set are assumed. Refer to "line number specification" in the list of general terms in Section 2 of Part II.)

```
System:    ANYREGN0000100 LINE 1
           ANYREGN0000200 LINE 2
```

```
ANYREG0000300 LINE 3
-
```

2. Display positions 2 through 4 of the data in lines 200 through 300.

```
User:     list 200(2),300(4)
System:   INE
          INE
          -
```

3. Display the value of the CLP.

```
User:     list clp
System:   0000400
          -
```

4. List the first and second characters of data in line 100.

```
User:     list 100(1),100(2)
System:   LI
          -
```

LL Command

The LL command is used to define the maximum length of any line to be written to the SYSOUT.

Operation	Operand
LL	LGH=,*TRUNCATE=,*RESET=

LGH
: decimal number, 1 or greater, defining the length of the ouput line which can be written to the SYSOUT. Any line greater in length, will be continued on the next line.

  Specified as: A decimal number from 1 to maximum physical line length for SYSOUT.

  System default:
  2741 -- 132
  Teletype -- 72
  3215/1052-7 -- 132

*TRUNCATE
: self-defining keyword. If TRUNCATE=Y is specified, then any output line greater than the specified 'LGH' value will not be continued on the next line. Any text in excess of the above LGH value will be deleted and not displayed.

  Specified as:
  Y = truncate output.
  N = do not truncate output.

  System Default: N

*RESET
: (see specification below)

  Specified as:
  Y = restore the original system default length and turn off truncate, if on.

N = nc effect.

System Default: N

Functional Description: The LL command can be used to lengthen and shorten the messages and data displayed by the system. On long lines, the user can also have the system delete any data over a specified length instead of writing continuation lines on the terminal.

The system defines a line as all output from one write request and not containing either a 'Newline' or a 'Carriage Return, Suppress' control character.

If the data contains a 'Newline' control character, then the system will treat each 'Newline' character as the end of one line of output and will write the data that follows on the next line.

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}] [,LIB=data definition name of library] [,VERID=version identification][,TSD={Y|N}][,PMDLIST={Y|N}] [,LISTDS={Y|N}][,LINCR=(first line number,increment)]

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 automatically prefixes 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 character string of from one to eight alphameric characters, the first of which must be alphabetic, can be specified. The object 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.

LIB
identifies the library in which the new object module is to be included. The user must either choose a library that does not contain any control section or entry point names identical to those in the output module or must rename the control section and entry point names in the output module during linkage editing.

Specified as: the data definition name of the library.

System default: the last-mentioned library is assumed (that is, the user library or a job library).

VERID
specifies the version identification to be assigned to the object program.

Specified as: from one to eight alphameric characters.

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.

Note: An ISD can be produced only if the source module contains an ISD.

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.

LISTDS
determines whether the user-requested listings from the language processors are to be placed in a list data set or placed directly on SYSOUT.

Specified as:

Y - place in list data set.
N - listings to SYSOUT.

System default: Y.

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-digit decimal numbers separated by a comma and enclosed in parentheses. The last two digits in each number must be zeros.

System default: (100,100).

Note: This operand is ignored when STORED=Y.

Functional Description: See "Language Processing," in Section 3 of Part II.

Caution: The output module from the linkage editor cannot be placed in the library specified in the input operand if that library contains modules whose control section or entry point names are identical to control section or entry point names in the output module.

The command is canceled if invalid operands are entered.

Example: The user wants to link-edit modules into an object module named ABCD. Conversationally, he enters all LNK operands and linkage-editor control statements from the terminal. The linkage editor takes default values for the remaining 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.

User: lnk abcd,n

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 searches the libraries on the task's current program library list to find the specified object module and loads the module. If the module is already loaded, no action is taken. 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.

When the LOAD command is issued with no operands, the user's module last referenced by one of the following commands is loaded: PLI, ASM, LNK, FTN, LOAD, UNLOAD, CALL with a module name specified, or an implicit call to a module.

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 (for example, 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, because only the module name can be found by the dynamic loader.

Example: Load module ABC, and all modules to which it implicitly refers.

User: load abc

The system loads ABC and all implicitly linked modules into virtual storage.

LOCATE Command

This command searches a region for a specified character string. LOCATE does not alter the referenced data.

Operation	Operand
LOCATE	[N1=starting position][,N2=ending position] [,STRING=character string]

N1

identifies a line, or the first of a series of lines, in the current region to be searched for STRING.

Specified as: a one- to seven-digit decimal 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 character of the record's key (position K), he can specify the starting position enclosed in parentheses, and immediately following the line number. The system normally begins the search with position K (the first character of the key); the first character of data is at position 1 and is specified as (1) following the line number. If the user wants to get to some other data character, he can specify its position as an absolute one- to four-digit decimal number. Any character between the first character of the key and the first data character can be referred to by a negative value enclosed in the parentheses. For example, the second character in the key for line 0000100 of a line data set is referred to as 100(-6).

System default: when N2 is specified, the value of the CLP is assumed. Otherwise, the entire data set or region is searched.

N2

identifies the last of a series of lines to be searched for STRING.

Specified as: a one- to seven-digit decimal 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. The ending character is included the command processing.

System default: when N1 is specified, it is the only line searched. Otherwise, the entire data set or region is searched.

STRING
designates the character string that is to be searched for (that is, the string is the "search argument"). Strings that are continued in the next line are not recognized.

Specified as: a normal or quoted string.

System default: CLP is set to the next line in the current data set or region. If N2 is the last line, CLP is set to N2 plus the value of INCR.

Functional Description: LOCATE searches the specified lines for the string (this search includes the region name and line number). 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 is issued without operands, the CLP is set to the line following the last line in the range specified (N2). If N2 is the last line, CLP is set to N2 plus the value of INCR.

The user is then prompted for a command.

Caution: A language-processing command (EDIT, PROCDEF, or PLI) must be invoked before the command is issued.

Examples:

1. The user wants to search the current region for the string ABC. (Note: A blank separates the last letter in LOCATE and the comma.)

   User: locate ,,'abc'
   System: 0000200 ABC WILL APPEAR IN SEVERAL LINES

2. The user wants to search lines 200 through 500.

   User: locate 200,500,abc
   System: 0000200 ABC WILL APPEAR IN SEVERAL LINES

3. The user restricts the search in example 2 to character positions 1 through 26 in lines 200 through 500. The system displays the physical line in which ABC is first found and prompts for a command. The LOCATE command the user enters is:

   User: locate 200(3),500(26),abc

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 data definitions (and I/O devices) used by the task.

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

Programming Notes: 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.

LOGON Command

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

Operation	Operands
LOGON	user identification[,password][,addressing][,charge number] [,control section packing][,maximum auxiliary storage] [,pristine][,user IVM code]

Notes: The LOGON command name must always be entered and operands must be entered in positional notation. Trailing commas need not be entered. If one or more operands is omitted, and a later operand is used, you must enter a comma for each omitted operand.

user identification
identifies the user to the system.

Specified as: the user identification assigned to the user when he was joined to the system.

password
specifies the user's assigned password.

Specified as: the password assigned to the user when he was joined to the system.

System default: in conversational mode, none, if the user has been assigned a password. In nonconversational mode, password is optional and will not be verified.

addressing
specifies the system addressing.

Specified as: 24- or 32-bit addressing.

System default: present system addressing.

charge number
specifies the charge number to be used if the installation has task accounting. This does not need to be the charge number assigned when the user was joined to the system.

Specified as: from one to eight alphameric characters.

System default: the charge number assigned when the user was joined to the system.

control section packing
specifies the type of control section packing to be provided by the dynamic loader.

Specified as:

A - all control sections will be packed.

P - only prototype control sections (PSECTs) will be packed.

O - only control sections having neither public nor prototype attributes will be packed.

X - all control sections except prototype control sections will be packed.

N - no control sections will be packed.

System default: N is assumed.

maximum auxiliary storage
specifies the expected maximum number of auxiliary storage pages required in the session.

Specified as: from one to five decimal digits.

System default: the lesser of either the system default established at system generation or the limit assigned to the user when he was joined.

pristine
indicates whether the user wants access to his previously defined defaults, synonyms, and PROCDEFs or all options or his USERLIB.

Specified as:

P - the user will be able to use only system-specified defaults, synonyms, or PROCDEFs; he can create synonyms, defaults, and PROCDEFs during this task, but he cannot add them to his profile for use in a subsequent task.

X - the user will be able to use only system-specified synonyms, defaults, and PROCDEFs: he cannot access anything in his USERLIB; he cannot create synonyms, defaults, or PROCDEFs, unless he first defines USERLIB.

System default: the user will have all of his previously specified defaults, synonyms, and PROCDEFs available to him, as well as all other members of his USERLIB.

user IVM code
indicates whether a user wishes to modify the contents of his user IVM.

Specified as:

Y - the user with U authority can modify the contents of his user IVM with a new set of modules.
N - the user cannot modify his user IVM.

System default: N.

Functional Description: The credentials the user enters (user identification and any of the operands required by your installation) are compared with the authorization data that identify him to the system. When any or all are not valid, the conversational user is prompted to enter all operands again. If the system responds with a question mark (?), the LOGON command was not recognized; the entire command must be reentered. When these credentials are valid, the task continues. LOGON calls ZLOGON before control is given to the user.

Programming Notes: LOGCN must precede any commands the user intends to issue. When the user turns on his terminal and dials the system, the system waits for the user to log on.

If the user has never been authorized to use the system, or 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.

   The system unlocks the terminal.

   User: logon frankdoe,mars7

   The system acknowledges that user has successfully logged on.

2. A nonconversational task is being started; the first prestored command in the nonconversational SYSIN data set is:

   ```
   LOGON FRANKDOE
   ```

LTDS (List TAPE Datasets) Command

This command will list the dataset name, file sequence number, and volume sequence number of all datasets on a tape created by the VT command.

Operation	(No Operands)
LTDS	

Functional Description: The LTDS command can be used to list all datasets on a tape created by the VT command. The tape must be previously defined by a ddef command with ddname of DDTVIN, and must be an unlabelled 9 track tape. Processing of datasets will begin with the file sequence number specified in the label operand of the ddef command and continue until EOT is detected.

Example: The user wants to list all datasets on tape TESTXX starting at file 2.

```
User:   DDEF DDTVIN,PS,TESTDSN,VOLUME=(,TESTXX),UNIT=(TA,9),-
        LABEL=(2,NL),DISP=OLD
        LTDS

System: VSN TESTXX,FSQ 0002.USERID**.DSFILE2
```

MCAST Command

This command alters the control characters in the user's Profile Character Switch Table (see Appendix C).

Operation	Operand
MCAST	[EOB=end of block character][,CONT=continuation character] [,CLP=break character] [,TRP=transient statement prefix character] [,RCC=concatenation character] [,SSM=system scope mask][,USM=user scope mask] [,KC=keyboard/card reader character] [,RS=carriage return suppression character] [,CP=command prompt string]

Note: All hexadecimal numbers must be enclosed in apostrophes and preceded by an X (as in X'62').

Since no system-supplied default values exist for these operands, we list the settings as they appear in the system-supplied version of the user profile.

EOB
specifies the end-of-block character.

Specified as: X'26'.

System default: X'26'.

CONT
specifies the command system continuation character.

Specified as: any single character or any hexadecimal number in the range X'00' to X'FF'.

System default: hyphen (X'60').

CLP
specifies the command system break character.

Specified as: any single character or any hexadecimal number in the range X'00' to X'FF'.

System default: underscore (X'6D').

TRP
specifies the transient statement prefix character.

Specified as: any single character or any hexadecimal number in the range X'00" to X'FF'.

System default: vertical stroke (x'4F').

RCC
specifies the concatenation character.

Specified as: any single character or any hexadecimal number in the range X'00' to X'FF'.

System default: colon (X'7A').

SSM
specifies the system scope mask.

Specified as: any single character or any hexadecimal number in the range X'00" to X'FF'.

System default: X'29'.

USM

specifies the user scope mask.

Specified as: any single character or any hexadecimal number in the range X'00" to X'FF'.

System default: X'29'.

KC

specifies the SYSIN keyboard/card reader character.

Specified as: the letter K, which tells the system to get input from the terminal keyboard; or the letter E, which tells the system to use the value of the SYSIN implicit operand. If SYSIN=K, input is from the keyboard; if SYSIN=C, input is from the card reader.

System default: E.

RS

specifies the carriage return suppression character.

Specified as: any single character or any hexadecimal number in the range X'00" to X'FF'.

System default: colon (X'7A').

CP

specifies the command prompt string.

Specified as: a string of from one to eight characters, or a hexadecimal number in the range X'00' to X'FFFFFFFFFFFFFFFF' (16 hexadecimal digits).

System default: an underscore, backspace, and a carriage return suppression character (X'6D167A').

Functional Description: The MCAST command replaces control characters in the user's task profile with the control characters specified as command operands. If no operands are entered, only those control characters are changed for which the user has defined default values (for example, DEFAULT CP=SIR).

To make these changes permanent, the user must follow the MCAST command with a PROFILE command in the same task.

Note: All of the control characters for the MCAST command are explained in Appendix C. Any unprintable hexadecimal values (for example, X'25') are ignored when included in the command prompt string.

Caution: Do not change the EOB character from the system-supplied value X'26'.

Examples:

1. The user wants to change his command prompt string to "SIR?", and his break character to the at sign (@):

   User: mcast cp=sir?,clp=@
   System: SIR?

2. The user wants to change the command prompt string to YES, and the continuation character to an asterisk (*); he wants to make these changes permanent in his user profile:

User: mcast cp=yes,cont=*
Sys,User: YES profile

3. The user wants to change his command prompt string to the word GO, underscored, and followed by a question mark (GO?). Since the back- space character (see Appendix C) is a hexadecimal number (X'16') with no printable value, the entire prompt string must be coded as a hexadecimal number.

User: mcast cp=x'c7d616166d6d6f'
System: GO?

Note that: X'C7' is a G; X'D6' is an O; X'16' is a backspace; X'6D' is an underscore; and x'6f' is a question mark.

MCASTAB Command

This command alters the translation tables in the user's task profile by replacing one or both with the replacement tables located by the labels SYSTRIN (for input) and SYSTROUT (for output).

Operation	Operand
MCASTAB	[INTRAN={N|Y}][,CUTRAN={N|Y}]

INTRAN
specifies whether to replace the current input translation table with the replacement table (SYSTRIN) or with the version that was current when the user logged on.

Specified as:

Y - replace with SYSTRIN.
N - replace with version current at LOGON.

System default: N.

OUTRAN
specifies whether to replace the output translation table with the replacement copy in SYSTROUT, or with the version that was current when the user logged on.

Specified as:

Y - replace with SYSTROUT.
N - replace with version current at LOGON.

System default: N.

Functional Description: MCASTAB replaces the input and output translation tables with the replacement versions located by the labels SYSTRIN (for input) and SYSTROUT (for output). The user can alter the replacement tables with the SET command; he can effect replacement with MCASTAB.

To make a change in the input translation table, the user alters the replacement version:

```
set systrin.(x'c1',1)=x'c2'
```

He specifies the location in the translation table of the character or function code he wants to change (X'C1' in this case); then he sets the location to the new value (X'C2' in this case). In this example, the

user has set the uppercase "A" to the value of "B". (Notice that the values are not swapped; both "A" and "B" have the value of "B".) Now, to make the change, the user issues the MCASTAB command, which replaces the current translation table with the altered version at SYSTRIN:

```
mcastab intran=y
```

The user can make the change permanent with the PROFILE command. This is, however, not recommended until the user has had some experience with changing the tables.

To reverse the changes, the user issues the MCASTAB command with the operand set to N; this replaces the current table (now it is SYSTRIN) with the version that was current when the user logged on. But, since the user lost the use of the "A" in the above example, he must log off and log on again to get to the original table since he is unable to enter MCASTAB successfully. The user should swap values wherever possible to retain the use of the entire character set. Had the user swapped the "A" and the "B" in the above example, he could then reverse the changes by issuing:

```
mcastab intran=n
```

Caution: Be careful not to lose the use of any important character by replacing it without choosing an alternate.

Programming Notes: At LOGCN, SYSTRIN and SYSTROUT always reflect the system-supplied translation tables (as shown in Appendix C). Whenever the user logs on, he can issue

```
mcastab intran=y,outran=y
```

to replace his translation tables with the system-supplied versions.

The user can also use the MCASTAB command to reverse any translation table changes made with the MCAST macro instruction.

Examples:

1. The user wants to swap the characters "A" and "B" in the input table:

```
Sys,User: set systrin.(x'c1',1)=x'c2'
Sys,User: set systrin.(x'c2',1)=x'c1'
Sys,User: mcastab intran=y
```

2. Now, the user wants to reverse the changes and get back to the system-supplied translation table:

```
Sys,User: mcastab intran=n
```

3. The user wants to convert the pound sign (#) to a backspace character for input as data; he wants to use the asterisk as the cancel character in place of of the pound sign:

```
Sys,User: set systrin.(x'7b',1)=x'16'  (translate # to a backspace
                                        character)
Sys,User: set systrin.(x'17b',1)=X'00' (remove the cancel function
                                        code (x'0c') from the #)
Sys,User: set systrin.(x'15c',1)=x'0c' (assign the cancel code to
                                        the asterisk)
```

Sys,User: mcastab intran=y (effect the replacement)

MODIFY Command

This command inserts, deletes, replaces, or reviews lines of a VISAM 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}] [,FTN={Y|N}]

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. If the VISAM data set has hexadecimal keys (rather than the usual EBCDIC key or line number), the data portion of each record in the data set must also be hexadecimal.

Specified as: a fully qualified data set name and (optionally) a 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. The review option cannot be used on a line that contains hexadecimal data.

Specified as: R.

System default: no review of records.

Note: The next four operands must all be explicitly entered as operands of MODIFY (or by the DEFAULT command), or all be omitted. When they are specified, MODIFY assumes that the data set being specified is not a line data set.

LRECL
designates the length, in bytes, of each fixed-length logical record.

Specified as: a decimal number. The maximum length for VISAM is 4000 bytes.

System default: 132.

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. The maximum key length is 244 bytes.

System default: 7.

RKP
specifies the displacement of the key field from the first byte of each logical record.

Specified as: a decimal number. The maximum key displacement is 4000 bytes.

System default:

4 is assumed if RECFM=V.
0 is assumed if RECFM=F.

RECFM
indicates the format of the data set records.

Specified as:

V - variable-length records.
F - fixed-length records.

System default: V.

Note: When the LRECL, KEYLEN, RKP, and RECFM parameters are specified, MODIFY assumes that the data set is not a line data set.

FTN
specifies that the MODIFY command is being executed to update an existing FORTRAN source data set (via card input) that was created using the FTN option of the DATA command or the DATASET card. When this option is specified, it is assumed that the card input is punched in keyboard format and the input is processed accordingly.

Specified as:

Y - this function is required.
N - the function is not wanted.

System default: N.

Functional Description: If LINENO=Y, the system asks the user for modifications by issuing a pound sign (#) and returning the carriage to the margin. Otherwise, there is no prompting, and the user can enter modifications after each carriage return. The user indicates his modifications by following these conventions:

1. Insert or replace a record.

   key,data

   key
   the key of the record to be inserted or replaced; for a line data set, this is the line number.

   Specified as: from one to seven digits.

   data
   the new data of the replacement or insertion record; a maximum of 120 characters is permitted in a line data set.

Note: Modifications to a VISAM data set with hexadecimal keys and data must be entered as keydata. Key and data must be specified as one hexadecimal string (for example, X% keydata).

2. Delete a record or a range of records.

D,key[,last key]

key
the key of a single record to be deleted or the first key of a range of records to be deleted.

last key
the final key of a range of records to be deleted.

3. Review a record or range of records (whether or not the review option is specified) without taking any other action with the records.

R,key[,last key]

key
the key of a single record to be reviewed or the first key of a range of records for review.

last key
the final key of a range of records to be reviewed.

The user indicates that he has completed his modifications by entering %E or by entering as the first character of a line a single break character followed by a command.

Note: The complete key must be given for a VISAM data set that is not a line data set.

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 accepts strings of EBCDIC representations of hexadecimal digits, converts them into machine representations of hexadecimal digits, and inserts them in a data set as directed by the user.

The EBCDIC string representing the hexadecimal data is entered in the 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 two EBCDIC characters in the string to one hexadecimal character, until the first nonhexadecimal character is encountered.

In performing the required conversion, the system checks to ensure 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 alphameric format are displayed at the terminal (REVIEW option), they will be lost in the transmission. There is no REVIEW option for the HEX option.

Caution: The DATA and MODIFY command names may be included in the records entered under a MODIFY command, but multiple break characters must be entered to end the DATA or MODIFY command in the data set. The first %E or single break character is interpreted as the end-of-input record for the current MODIFY command.

Programming Notes: To save processing time, the user should enter his modifications in sequence, starting with the lowest line number.

By making a series of insertions, the issuer 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, Section 1, under "Entering Command Statements" in conversational mode).

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, RKP=3, fixed-length records, and the record length. 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. (If the two lines should not be run together, use a blank space before the hyphen.) The hyphen does not become part of the record; 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.

Examples:

1. The user wants to delete lines 107 through 195 and replace line 107 in a line data set ASET. Review is not requested. (LIMEN=I.)

```
User:       modify source.aset
System:     ENTER MODIFICATIONS
Sys,User:   #
            d,107,195
            #
            107,x=a**2.0
            #
            108,write(2,5)x
            #
            109,end
            #
            _ftn aset,y
```

   To end modifications, the user enters a break character and a command after the system prints out the pound sign and returns the carriage.

2. The user wants to delete line 4900 and insert a new line at line number 5450 in his partitioned line data set AB12.CA(V8). He requests review.

```
User:       modify ab12.ca(v8),r
System:     ENTER MODIFICATIONS
Sys,User:   #
            d,4900
System:     00004900X=(X=C) (prints out the deleted line for review)
Sys,User:   #
            5450,j=j+1
System:     00005450j=j+1 (prints out the inserted line for review)
Sys,User:   #
            %e
System:     _
```

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:     ENTER MODIFICATIONS
Sys,User:   #
            12300,somer=b/c
System:     12300SOMER=B-C (reviews old line)
Sys,User:   #
            14350,i=12
System:     14350I=12 (prints out the inserted line for review)
Sys,User:   #
            %e
System:     _
```

4. The user wants to create a new VISAM data set named QUIK4. Records are to be 80 bytes and 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:     PROCEEDING:  DATA SET OR MEMBER WILL BE CREATED ENTER
            MODIFICATIONS
Sys,User:   #
            ab00411 spring,retaining
            #
            ab00412 spring,guide
            #
            ab00413 clip,retaining spring
            #
            ab00414 widget,silverplated
            #
            %e
System:     _
```

5. The user wants to create a new line data set named DISSMAL. Review is not wanted. (LIMEN=I.)

```
User:       modify dissmal
System:     PROCEEDING:  DATA SET OR MEMBER WILL BE CREATED
            ENTER MODIFICATIONS

Sys,User:   #
            100,ald dc f'875'
            #
            200,smel dc f'5280'
            #
            300, dc f'6793'
            #
            400, dc f'557'
            #
            %e
System:     _
```

NUMBER Command

This command renumbers a line or a range of lines within the current region.

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

N1
identifies the line or first of a range of lines to be renumbered.

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

LAST - last line in the current region or data set.

System Default: N1 is set to the value of CLP. If N2 is defaulted, N1 is set to the first line of the current region or data set.

N2
identifies the last of a range of lines to be renumbered.

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

LAST - last line in the current region or data set.

System default: N1 is assumed if it was specified. Otherwise, N2 is set to last line in the region or data set.

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

Specified as: from one to seven decimal digits. The value must not be less than N1.

System default: The value of N1 or its default value.

INCR
specifies the increment between the lines to be renumbered.

Specified as: from 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 that the increment has been furnished by the system.

System default: the difference between the base and the line number following N2 is divided by the number of lines to be renumbered. The increment is then determined in this manner:

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

Note: If all operands are defaulted, these values are assumed:

N1=First line of region
N2=LAST
NBASE=100
INCR=100

Functional Description: Renumbering does not change the sequence of lines or affect the region name prefixed to line numbers.

When all operands are defaulted, the entire data set or region is renumbered.

Upon completion of this command, the CLP is positioned to N2 plus the value of INCR or to the line number after N2, whichever is less.

If the NBASE is 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.

Caution: A language-processing command (EDIT, PROCDEF, or PLI) must be invoked before the command is entered.

If the NUMBER command is not allowed to run to completion, the user may lose data.

Examples:

1. number 103,290

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

Since NBASE 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. As the quotient is 33 (297 divided by 9 = 33), the increment is 20.

2. number 17,,22

Original Sequence	Resulting Sequence
AR0000010	AR0000010
AR0000017	AR0000022
AR0000035	AR0000035

3. number 912,1000

Original Sequence	Resulting Sequence
AR0000900	AR0000900
AR0000912	AR0000912
AR0000915	AR0000932
AR0000916	AR0000952
AR0000917	AR0000972

AR0000918	AR0000992
AR0001000	AR0001012
AR0001050	AR0001050

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

Original Sequence	Resulting Sequence
M0000001	M0000001
M0000005	M0000006
M0000008	M0000019
M0000009	M0000032
M0000100	M0000100

5. number 100,200

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

ODC Command

This command converts an OS text deck into a TSS object module and stows the module into the highest joblib.

Operation	Operand
ODC	ODCMOD=module [,CDCPLI=Y|N] [,ODCERASE=Y|N]

module
the name of a test dataset which is a generation data group dataset of the form LOAD.module(0)

ODCPLI
tells the command whether the text deck was produced by the PL/I Optimizer. This operand can be defaulted if deck was produced by COBOL, FTHH or HASM.

ODCERASE
signifies whether the command should erase the LOAD.module(0) dataset following completion of the conversion.

Note: When you convert a PL/I module, ODC checks the value of PLIPACK, in your user profile. If PLIPACK=N, one or more pages are assignedto each CSECT. If PLIPACK=Y, CSECTs are packed. Packing consists of combining CSECTs into contiguous storage, retaining doubleword boundaries for CSECT origins. The name of the initialization CSECT is retained as a CSECT name, and other CSECT names are transformed into entry-point names. In effect, the CSECTs are combined into a single CSECT. If PLIPACK=P, ODC packs all CSECTs except static external CSECTs that have the TSS COMMON attribute, or are more then 4096 bytes long. This is generally more efficient than PLIPACK=Y, since COMMON CSECTs are null CSECTs and they are mapped onto external storage only if they are packed. The IBM-supplied default for PLIPACK is P.

OSDD? Command

This command will list to SYSOUT all filedeffed datasets, indicating OS ddname and TSS dataset name.

Operation	Operand
OSDD?	

Note: If a TSS dataset is datadeffed, but not filedeffed, it will not appear in this list.

OSRUN Command

This command allows the user to execute the output of a program product under TSS, using the Program Product Language Interface (PPLI).

Operation	Operand
OSRUN	module[,'parm']

module
specifies the name of the program to be run using the PPLI. It must have been assembled/compiled using the PPLI.

parm
this represents a value that will be passed to the program being run. Note that to pass an argument to a PL/I Optimizing Compiler main procedure the argument must be preceded by a slash.

Example: osrun PL130,'¢234567890'

Functional description: This command will invoke a routine which sets up the PPLI environment (i.e., issues SIR's for SVC's and initializes required control blocks). It then invokes the module specified, and upon return, deactivates the PPLI environment.

PC? Command

This command presents the name, access, and, for shared data sets, the owner's identification of one or more cataloged data sets.

Operation	Operand
PC?	NAMES=(data set name|(data set name[,...]))

Note: Managers and administrators should see Manager's and Administrator's Guide for special operands.

NAMES
identifies one or more cataloged data sets. If a partially qualified data set name is specified, each data set with the same qualification is presented.

Specified as: one or more fully or partially qualified data set names. When two or more data set names are specified, they must be enclosed in parentheses.

System default: every data set in the user's catalog is presented.

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

- Name - the name of the data set is given.
- Access - if the data set is owned by the user, the owner access is given; if the data set is owned by someone else, the sharer access is given.
- Ownership - if the data set is owned by someone else, the user identification of the owner is given.

In conversational mode, the information is presented at the terminal. In nonconversational mode, the information is printed in the SYSOUT data set.

Programming Notes: Presentation can be terminated in conversational tasks at any point by pressing the ATTENTION key. The DSS? command can be used for more thorough information about cataloged data sets.

Examples:

1. The user wants the names of all his data sets.

```
User:    pc?
System:  DATA SETS IN CATALOG WITH QUALIFIER NICHOLAS
         NICHOLAS.USERLIB,   ACCESS: RW
         NICHOLAS.NICHOLAS.TEST,   ACCESS: RW
         NICHOLAS.TA000304.SOURCE.SINGLE,   ACCESS: RW
         NICHOLAS.TA000307.SOURCE.FCB,   ACCESS: RW
         NICHOLAS.TA000310.SOURCE.DOUBLE,   ACCESS: RW
         NICHOLAS.TA000313.TMPDBL,   ACCESS: RW
```

2. The user wants the names of all his data sets with the qualification J.S.B..

```
User:    pc? j.s.b.
```

The system presents the information for all the data sets with the qualification j.s.b.

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} [,USERID={(user identification[,...])|*ALL}] [,ACCESS={R|RO|RW|U}]

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.)

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.

ACCESS
designates the access qualification for users sharing the data sets.

Specified as:

R - restricts access; sharing access that was previously permitted is withdrawn.

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: If a list of sharers is being updated, the access of the last sharer in the list is assumed. If a new list of sharers is being created, U 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.

Cautions: If a sharer erases a data set to which he has been given unlimited access, the entry of 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, the original data set definition is not changed (for example, it indicates private ownership). If a second user issues a SHARE command for the data set, the owner must release the existing data set definition before the sharer can use it.

Programming Notes: The designated sharers must issue SHARE commands to link their catalog entries to the owner's. The sharers can reference the data set under the owner's catalog entry only after the PERMIT command has been issued.

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.

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. 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,*all,r
   ```

   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:

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

2. 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,(joseph24,henry24a),ro`

   The system enters a list of sharers in owner's catalog.

3. 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,(joseph24,henry24a),rw`

   The system updates the sharing list.

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

   User: `permit userlib,(jbrown#1),ro`

   The system enters the sharing list in owner's catalog.

PLI Command

This command invokes the PL/I compiler and compiles a source program module.

Operation	Operand
PLI	[NAME=module name][,PLICPT=compiler option list] [,PLCOPT=language controller options] [,SOURCEDS=source data set name] [,MERGELST=converter input list] [,MERGEDS=converter input data set] [,MACRODS=intermediate data set name] [,EXPLICIT=external names to be changed] [,XFERDS=transfer vector data set name]

NAME

The name by which the program will be known. It consists of one to eight alphameric characters, the first of which is alphabetic. If the name is omitted, PLC assumes that it is identical to the name of the source data set if that is in the correct form. If neither NAME nor SOURCEDS is provided, no compilation takes place and PLC proceeds to process the merge list or go on to the next set of PLI

parameters. See PL/I Programmer's Guide for a complete list of naming rules.

Note: In nonconversational mode, PL/I source statements can follow the PL/I command in the input card deck. See PL/I Programmer's Guide for further information.

PLIOPT
The list of options to be used by the compiler. It is considered to be one parameter, and the list of compiler options following the equal sign in the PLIOPT parameter must therefore be enclosed in parentheses unless only one value is given; the separate options are separated by commas. The compiler options are described in Appendix I.

PLCOPT
A list of options external to the PL/I compiler that effects the compilation's progression through TSS. These options must be enclosed in parentheses unless only one value is given. The options and the standard default for each are shown in Table 18.

Table 18. PLCLPT options and system defaults

PLC Option	Standard Default
NOPRINT|PRINT|PRERASE	NOPRINT
DIAG|NODIAG	DIAG
NOCONT|CONT	NOCONT
LISTDS|LISTOUT	LISTDS
NOCONV	
LIMEN=	system defaults
BREVITY=	

The PLC options shown in Table 18 are defined as follows:

NOPRINT or PRINT or PRERASE
this option specifies whether the listing data set produced by the compiler is to be printed on a high-speed printer. NOPRINT indicates that the data set is not to be printed as a part of the compilation. You can at some later time issue a PRINT command directly as follows:

```
PRINT LIST.XXX(0),,,EDIT
```

where XXX is The module name given in the NAME operand. PRINT indicates that PLC should issue the print request automatically. PRERASE indicates that PLC should cause the data set to be printed and erased after printing; this is equivalent to:

```
PRINT LIST.XXX(0),,,EDIT,ERASE
```

Normally PLC does not issue any print requests.

If LISTOUT is specified, the data normally written into the list data set is directed to SYSOUT and no print request is appropriate. In this case the value of this print option is forced to NOPRINT under any circumstances.

DIAG or NODIAG
this option specifies whether diagnostics are to be directed to SYSOUT or not. (This option only has meaning if LISTDS is specified. If LISTOUT is specified, then all compiler diagnostics

appear on SYSOUT as a part of the listing data.) If DIAG is specified, then the diagnostics that will appear on SYSOUT are controlled by two command-system defaults, LIMEN and BREVITY, which control the severity and length of the PL/I diagnostics selected for printing on SYSOUT. The LIMEN and BREVITY operands of the PLI command are explained later in this section.

The format of the diagnostic message is:

```
x IEMnnnnI  statement no.   line no.  text
```

where x is the severity of the diagnostic and nnnn is the diagnostic number. For example:

```
S IEM0182I   15     1600 TEXT BEGINNING 'KEYFROM CK' SKIPPED IN
                         OR FOLLOWING STATEMENT NUMBER 15
```

If no option is specified, then DIAG is assumed.

NOCONT or CONT
specifies whether additional programs are to be compiled before return to the command system. NOCONT implies that there is no continuation of compilation. This is assumed if no value is specified. If CONT is specified, then PLC prompts for a new module name with PLI on a new line if none was given in the original PLI command. To end the prompting, enter an underscore with a command, or default by pressing the RETURN key.

This CONT event can be repeated as often as necessary.

LISTDS or LISTOUT
this option allows you to choose whether a separate data set should be constructed by the PL/I compiler for the computer listing or not. This is the default value specified explicitly by LISTDS. LISTOUT implies that a separate listing is unnecessary and that the listing output can be placed in SYSOUT. Particularly in nonconversational environment, the use of the SYSOUT data set is more efficient. Since in nonconversational operation the SYSOUT data set is automatically printed, the number of print requests is reduced as well as the overall load on the system.

In a conversational environment, placing the listing data on SYSOUT means typing this data on the terminal. Only in most urgent circumstances should you consider this alternative.

NOCONV
this option allows you to specify that no compilation is to occur. If NOCONV is selected, the MERGELIST operand should contain the names of the modules that are to be put into the transfer vector data set.

LIMEN=
LIMEN is the operand name in the user profile for message-severity codes. It controls the severity of diagnostic messages printed on SYSOUT. If specified in the PLI command, LIMEN applies only to PL/I diagnostics. (See DIAG, above.) If LIMEN is not specified, the current value in the system profile is used.

LIMEN Value	Lowest Level Diagnostic Issued
I (information)	Warning messages
W (warning)	Serious error messages
X (serious error)	Termination error message
T (termination error)	None is shown

BREVITY=
BREVITY is the operand name in the user profile for message-length codes; it controls the length of diagnostic messages printed on SYSOUT. If specified in the PLI command, it applies only to PL/I diagnostics. (See DIAG, above. If not specified, the current value in the system profile is used.

BREVITY Value	Output
M (message ID)	Message ID only
S (standard)	Full text of message
E (extended text)	Full text of message
T (standard, no ID)	Full text of message without message ID
X (extended, no ID)	Full text of message without message ID

Note: Both LIMEN= and BREVITY= can be followed by only one character. If the equal sign is not the next-to-last character, the option is ignored. Thus:

LIMEN=I is valid

LIMEN=INFO is invalid because more than one character follows the equal sign.

LIMEN= I is invalid because there should be no space after the equal sign.

SOURCEDS
the fully qualified name of the data set from which the PL/I source statements are to be obtained. Any valid line data set is allowable. Examples:

1. ABLE
2. A.B.C.D.
3. A.B(C)
4. A.B.G0000V00
5. A.B(0)
6. A.B(0) (C)

If the NAME operand is omitted, the SOURCEDS name is used as the name of the object module. Therefore, if the NAME operand is omitted and a TSS executable object module is to be generated, the source data set must not be in the last-defined job library, since the object module will be stored in that library. TSS does not allow a library to contain duplicate entry names.

If SOURCEDS is omitted, the name assumed for the source data set is SOURCE.XXX, where XXX is the value you gave for the NAME operand.

If neither NAME nor SOURCEDS is given, it is assumed that no compilation is to take place for this iteration of PLC. Other functions involving ODC may be involved. The system default for SOURCEDS is a string of blanks.

MERGELST
the names, separated by commas, of previously compiled modules to be converted by ODC for execution with the module being compiled. Each of these modules should still exist as data sets named IOAD. XXX(0), where XXX is the name given by you, or by default, in the

NAME operand. (Intially, the compiler creates all modules as LOAD.XXX(0) data sets. You should not erase these data sets until you are sure that you have all the needed copies of the converted object module.) Modules that have been stored in job libraries after processing by ODC cannot be used in a merge list.

If NOCONV is specified as a PLC option, this operand must contain the names of the modules that are to be transformed. You should keep the LOAD data sets if complete module refreshment is desired. Otherwise, you are not required to keep the LOAD data sets for reconversion.

If the MERGELST operand is omitted but the LOAD option is indicated in the PLIOPT list, the PL/I compiler still generates a merge list containing the name of the compiled program.

MERGELST is similar to the NAME cards generated by OBJNM=aaaaaaaa in PL/I. The merge list can be a single program name:

```
BAKER
```

or a list of program names enclosed in parentheses:

```
(FOX, GEORGE, HOW)
```

The list must not exceed 253 characters, including blanks and commas.

Duplicate program names in the list cause reprocessing of those programs. The only penalty is in terms of added processing time.

If no value is supplied for MERGELST, then a null string is assumed initially.

MERGEDS
: allows you to name a data set as the source of the merge list. This can be in lieu of MERGELST or a supplement to it. If this data set is VS or VT, it is assumed that each record contains from 0 to 15 program names separated by commas. As anywhere else, spaces are immaterial. The PLC and ODC assume that all program names in the MERGEDS for which a LOAD.XXX(0) data set exists are tobe combined into a single JOBLIB. Duplicate names cause duplicate processing, but otherwise do not hurt.

  If the data set is a VP data set, then it is assumed that all the member names for which a LOAD.XXX(0) data set exists are to be combined into a JOBLIB. If the current active JOELIB has the same name as MERGEDS, then all modules in the POD for which a PL/I LOAD.XXX(0) data set exists are to be reprocessed.

  If no value is supplied, no data set is assumed for MERGEDS.

MACRODS
: is the data set name to be associated with the intermediate text. If no name is given and either CHAR48 or MACRO options are specified, the compiler creates a data set named:

```
MAC.name(0)
```

  Where 'name' is the user-supplied object module name. This data set is normally erased when the compilation is completed. If you specify a value for MACRODS, that name is used instead of MAC.name(0) for the data set, and it is retained permanently with a compiler-generated source margin of 2 to 72. If a value is given for

MACRODS but neither CHAR48 nor MACRO is specified, the value is ignored and does not contribute to or hinder the compilation.

Note: When using this data set for recompilation, a source margin of 2 to 72 must be specified in the SORMGIN option of the PLI command's PLIOPT parameter.

EXPLICIT
specifies the external names to be changed and put into a transfer vector data set.

Specified as:

name - the external name.
(name+[,...]) - list of external names.
*ALL[(name+[,...])] - all external names that do not begin with IHE except the names listed in parentheses

Programming Notes: The system default, MAP, reports to the user the results of name changes to REFs. The default, N (no report), can be changed to Y (report) by issuing the DEFAULT command.

The padding character that is used when the external names are changed is the symbol @. This character can be changed to any alphabetic character or to the symbols # or $ by using the DEFAULT command, with PADCHAR specified as the operand.

The system default value, UPDTXFER, if entered as UPDTXFER=Y, specifies that new names can be entered into the transfer vector data set. If entered as UPDTXFER=N or if defaulted, no names can be entered into the data set.

XFERDS
is the name of the transfer data set that will be created. If this operand is omitted, there will be no transfer data set. An existing data set specified on this operand will be updated, and if the data set does not exist, it will be created.

Programming Notes: The system default value, PLIPACK, is checked to determine what type of CSECT packing will be done. The options are: (1) Y, all CSECTs of the input module will be packed; (2) P (the default), there will be partial packing of CSECTs; and (3) N, no CSECT packing will be done.

The system default value, REJMSG, is used to override the output of rejection messages by the loader. The options are: (1) N (the default), no overriding; and (2) Y, override.

You can change the initial settings of PLIPACK and REJMSG. Issue a DEFAULT command with the new value before issuing the PLI command.

PLIOPT Command

This command will invoke the PL/I Optimizing Compiler program product using the Program Product Language Interface.

Operation	Operand
PLIOPT	NAME=modulename[,OSOPTS=(opt1,opt2,...)]
	[,SOURCEDS=sourcedsname]

NAME
identifies the name by which the object program will be known to TSS. It consists of one to eight alphameric characters, the first of which is alphabetic. If the SOURCEDS option is not specified, there must exist a dataset called SOURCE.name which is assumed to be the source program to be compiled.

OSOPTS
specifies a list of OS options to be in effect during the compilation. Not all options are applicable in TSS.

Compiler Option	Abbreviated Name	TSS Default
AGGREGATE|NOAGGREGATE	AG|NAG	NOAGGREGATE
ATTRIBUTES|NOATTRIBUTES	A|NA	NOATTRIBUTES
CHARSET([48|60] [EBCDIC|BCD])	CS([48|60][EB|B])	CHARSET(60EBCDIC)
COMPILE|NOCOMPILE [(W|E|S)]	C|NC [(W|E|S)]	NOCOMPILES(S)
CONTROL('password')	----	----
COUNT|NOCOUNT	CT|NCT	NOCOUNT
DECK|NODECK	D|ND	NODECK
DUMP|NODUMP	DU|NDU	----
ESD|NOESD	----	NDESD
FLAG [(I|W|E|S)]	F [(I|W|E|S)]	FLAG(I)
GONUMBER|NOGONUMBER	GN|NGN	NOGONUMBER
GOSTMT|NOGSTMT	GS|NGS	NOGOSTMT
IMPRECISE|NOIMPRECISE	IMP|NIMP	NOIMPRECISE
INCLUDE|NOINCLUDE	INC|NINC	NOINCLUDE
INSOURCE|NOINSOURCE	IS|NIS	INSOURCE
LINECOUNT(n)	LC(n)	LINECOUNT(55)
LIST [(N,M)]|NOLIST	----	NOLIST
LMESSAGE|SMESSAGE	LMSG|SMSG	LMESSAGE
MACRO|NOMACRO	M|NM	NOMACRO
MAP|NOMAP	----	NOMAP
MARGINI('c')|NOMARGINI	MI('c')|NMI	NOMARGINI
MARGINS(m,n [,c])	MAR(m,n [,c])	MARGINS(2,72,0) or MARGINS(10,100,0)
MDECK|NOMDECK	MD|NMD	NOMDECK
NAME('name')	N('name')	----
NEST|NONEST	----	NONEST

NUMBER|NONUMBER	NUM|NNUM	NONUMBER
OBJECT|NOOBJECT	OBJ|NOBJ	OBJECT
OFFSET|NOOFFSET	OF|NOF	NCOFFSET
OPTIMIZE(TIME|0|2) NOOPTIMIZE	OPT(TIME|0|2)|NOPT	NOOPTIMIZE
OPTIONS|NOOPTIONS	OP|NOP	OPTIONS
SEQUENCE(m,n)|NOSEQUENCE	SEQ(m,n)|NSEQ	NOSEQUENCE
SIZE([-]YYYYYYYY| [-]YYYYYK MAX)	SZ[-]YYYYYYYY| [-]YYYYYK MAX)	SIZE(MAX)
SOURCE|NOSOURCE	S|NS	SOURCE
STMT|NOSTMT	----	STMT
STORAGE|NOSTORAGE	STG|NSTG	NCSTCRAGE
SYNTAX|NOSYNTAX [(W|E|S)]	SYN|NSYN [(W|E|S)]	NCSYNTAX(S)
TERMINAL [(opt-list)]| NOTERMINAL	TERM [(opt-list)]| NTERM	NOTERMINAL
XREF|NOXREF	X|NX	NOXREF

See Appendix L and OS PL/I Optimizing Compiler Programmer's Guide for further information.

SOURCEDS
specifies the name of the input data set to be compiled.

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?	[PODNAME=data set name][,DATA=Y][,ALIAS=Y] [,MODULE={module name|*ALL}]

PODNAME
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.

System default: USERLIB.

DATA
specifies that the system and user data (if any) 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 are defined by the user. Similarly, only certain system data (25 hexadecimal digits) can be printed.

Specified as: Y.

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

ALIAS
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: Y.

System default: the members' aliases are not listed.

MODULE
identifies the module (member name) for which information associated with that module will be printed. The information consists of the module's version ID and the name, version ID, attributes, and external references for each CSECT within the module. The CSECT entry points are provided by the ALIAS option.

Specified as:

module name - information is provided for the specific module.

*ALL - information is provided for all modules in the data set.

System default: no module information is printed.

Functional Description: If a VPAM data set is a program library (for example, user library 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 linkage 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 (for example, entry point names), the system also defines a number of aliases for the module (for example, all external symbol definitions except those named COMMCN 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. Additional information describing the version and external references of a module is contained in the program module dictionary (PMD) and is available to the user via the MODULE option.

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.

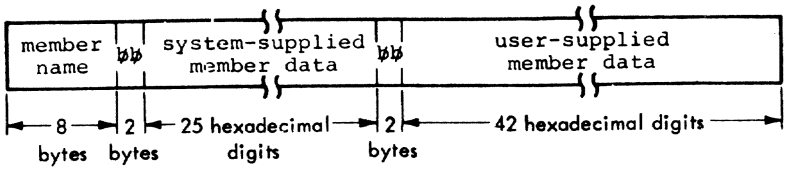
Programming Notes: The POD? command can be used to examine information pertaining to the members of any cataloged VPAM data set that a user owns or shares.

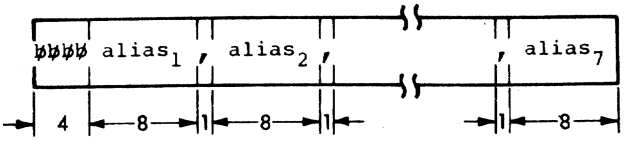
The conversational user may terminate the printout at any time by pressing the ATTENTION key.

The format of the information sent to SYSOUT after the POD? command is executed is shown in Figure 2.

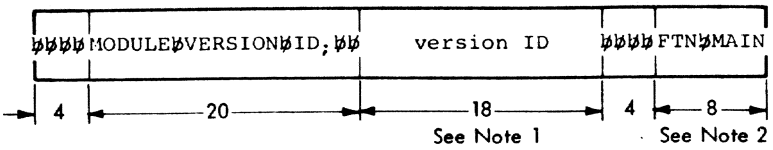
Example: The user wants to obtain a listing of the modules presently in his user library.

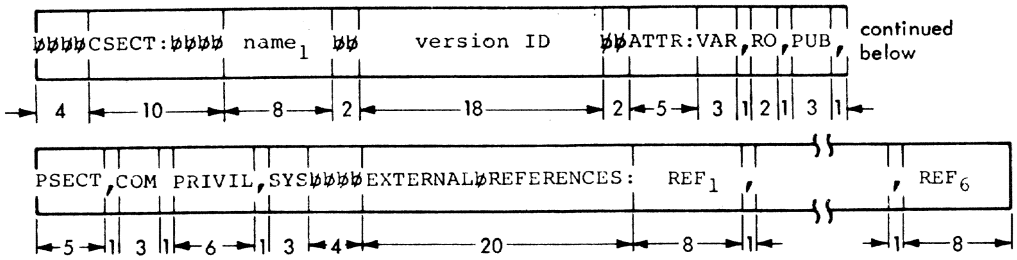
```
User:     pod? userlib,y,y
System:
 SYSPRX     200360003000100000000000
 JJCBHELP   2002E000300100000000000000   000000000000164000100000000200000000000000
     PSCNDX  ,RDINDX  ,GOINDX
 NIP        2001F000300100000000000000   000000000000D000010000000010000002000000
     PIN
 TWOCOL     2000C000300100000000000000   000000000000035C000100000000200000000000000
     TWOCOL#C,TWOCOL#P
```





If there are more than seven alias names, additional lines are printed that contain up to seven aliases each.





If there are more than six references, additional lines are printed that contain up to eight references each.

See Note 3

Note 1. The module version ID may appear in either of two formats, depending on the manner in which it was originally specified. One format consists of eight EBCDIC characters, and the other consists of a date and time in the format MM/DD/YY/bbHH:MM:SS (for example 07/22/71 12:15:08).

Note 2. If this is a FORTRAN module, this eight-byte field contains the words FTN MAIN or FTN SUBR.

Note 3. The CSECT name, version ID, attributes, and external references are repeated for each CSECT within a module. The attributes and external reference portions only are provided if at least one attribute or one external reference is applicable. Only the applicable attributes are provided without superfluous commas. The attributes that may be associated with a CSECT are Var (variable-length), RO (read-only), Pub (public), PSECT (PSECT), Com (common), Privil (privileged), and Sys (system).

Figure 2. Format of output from the POD? command for each member

POST Command

See "DISABLE, ENABLE, POST, and STET Commands."

PRINT Command

This command schedules a job to print a data set on a high-speed line printer.

Operation	Operand
PRINT	DSNAME=data set name[,STARTNO=starting position] [,ENDNO=ending position] [,PRTSP=EDIT|1|2|3| [,HEADER=H] [,LINES=lines per page] [,PAGE=P] [,ERASE={Y|N}][,ERROROPT={ACCEPT|SKIP|END}] [,FORM=paper form][,STATION=station id]

Note: System programmers should see System Programmer's Guide for specialized operands.

DSNAME
identifies the data set that is to be printed; VAM data sets must be cataloged; BSAM data sets 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: from one to six decimal digits.

System default: printing starts with the first byte of each record.

Note: In a VISAM line data set with no regions, the data begins in position 9.

ENDNO
specifies the byte position at which printing is to stop for each data set record; this end byte will be printed.

Specified as: from one to six decimal digits.

System default: printing continues to the last byte of each logical record or until the printer line length is reached, whichever occurs first. (The maximum printer line length is 132 characters.)

PRTSP
specifies the number of spaces to be skipped between lines.

Specified as:

EDIT - line spacing is controlled by a character in the first byte position of each logical record. The control characters may be either a FORTRAN control character (defined by American National Standard FORTRAN, ANSI X3.9-1966) or machine code (see Appendix D), 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.

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 is printed.

LINES
indicates the number of lines to be printed on a page.

Specified as: from one to four decimal digits; 9999 is the maximum.

System default: 54 lines are printed on each page.

PAGE
specifies that pages are to be numbered.

Specified as: P.

System default: no page numbers are assigned.

ERASE
specifies that the cataloged data set is to be erased from the catalog after the printing operation is finished.

Specified as:

Y - erase
N - save

System default: no erasure is 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 is accepted.
SKIP - error record is skipped.
END - print operation is terminated.

System default: END is assumed.

FORM
designates the identification of the desired combination of paper forms, print chain, carriage control tape, etc. See the installation manager for acceptable values.

Specified as: from one to six alphameric characters.

System default: the installation's standard printer form, chain, etc., is used (as specified at system generation).

STATION
indicates the RJE station identifier to which output is to be sent.

Specified as: up to eight alphameric characters.

System default: ID from Task Common is used.

Note: This parameter can be specified only if the user was assigned this capability when he was joined to the system.

Functional Description: PRINT assigns the request to an independent nonconversational task, to which the system assigns a BSN for possible reference by the user. The specified data set is printed as it appears. Invalid print characters appear as blanks in the output. Data set records containing a read error (or an invalid control character, when the EDIT option is used) are 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 includes that number in its instructions to the system operator when the printer is readied for operation.

The data set name specified for a BSAM data set 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: When the user issues PRINT for a BSAM data set that is defined in his task, the data set definition is released, and the data set is disconnected from the user's task.

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-U) format records.

PRINT should not be used for an uncataloged data set that is awaiting bulk I/O, as PRINT automatically erases an uncataloged data set.

Programming Notes: The user may use the BSN to identify his task when using the CANCEL command.

To print an unlabeled tape, the user must precede the PRINT command with a DDEF command and these operands:

```
DDNAME=name,PS,DSNAME=dsname,DCB=(RECFM=format,LRECL=length,
```

```
BLKSIZE=block size,DEN={0|1|2},TRTCH={C|E|T}),UNIT=(TA,tape type),

VOLUME=(,volume serial number),LABEL=(,NL),DISP=OLD
```

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:  BSN=0231
```

PRMPT Command

This command allows a user to manipulate and use the message file.

Operation	Operand
PRMPT	MSGID=message identification [,INSERTn=inserted character[,...]]

MSGID
: the identification of some message in the message file (SYSMIF). If identification is less than eight characters, it is padded with blanks on the right. If it is greater than eight characters it will be truncated.

INSERTn
: variable text to be inserted in the text of the message. The maximum text that can be inserted in a single insertion is 40 characters. If no inserted text is given when the message text expects some, the location of the inserts is indicated with three asterisks.

Functional Description: The PRMPT command can generate messages from the message file so that they can be examined and comprehended. This allows users to display the standard and extended forms of the message easily. The command also allows substitution of inserts into the text of messages and, with DISPLAY, allows messages to be issued from command procedures (PROCDEFs).

Example: The user has a message in his own message file that reads:

```
Z0065 LITHUANIA $01 SWING $02
```

He can issue this message with the PRMPT command as:

```
User:    prmpt zoo65
System:  Z0065 LITHUANIA *** SWING ***
User:    prmpt zoo65,does
System:  Z0065 LITHUANIA DOES SWING ***
User:    prmpt zoo65,can,too
System:  Z0065 LITHUANIA CAN SWING TOO
```

PROCDEF Command

The PROCDEF command defines a user-written command procedure that consists of other commands.

Operation	Operand
PROCDEF	NAME=procedure name[,PROLIB=data set name]

NAME
designates the name to be assigned to the command procedure.

Specified as: from one to eight characters. . This name must not contain embedded blanks, commas, semicolons, equal signs, or apostrophes.

PROLIB
specifies the library in which the PROCDEF is stored.

Specified as: the name of a VPAM data set. If the specified data set does not exist, it will be created. The PROCDEF is stored in the SYSPRO member of the data set.

System default: USERLIB.

Functional Description: When the user enters the PROCDEF command and operand, the text editor is invoked to monitor PROCDEF processing. The user can use all of the text-editing facilities during command creation. Unless the user suppresses line number prompting (DEFAULT LINENO=N), the system prompts him to enter data by issuing line numbers. For a new procedure, the system issues a line number with the value of BASE (100 is the default value for BASE). For an existing procedure, an underscore is issued; CLP is set to the first line in the PROCDEF.

Programming Notes: For a detailed discussion on defining command procedures, see "Command Procedure" in Section 4 of Part II.

If you specify a data set, other than USERLIB, in which to store the PROCDEF, you cannot execute that PROCDEF until you either redefine the data set as USERLIB or store the PROCDEF in USERLIB.

Example: If COPYCAT is the name of the command procedure being defined, the user enters:

```
procdef copycat
```

and the system replies:

```
0000100
```

If COPYCAT had been defined previously, the system's prompt would have been:

```
-
```

PROFILE Command

This command causes the task profile to replace the user profile in USERLIB.

Operation	Operand
PROFILE	[CSW={N|Y}]

CSW
specifies whether the command symbols are to be saved with the task profile.

Specified as:

Y - the command symbol definitions are saved with the task profile.
N - the command symbol definitions are not saved with the task profile.

System default: N.

Functional Description: When a PRCFILE command is issued, the task profile is written into the user library, replacing the previous version, and remains unchanged until another PROFILE command is issued or until the USERLIB (SYSPRX) is erased. As a result, values entered by DEFAULT or SYNONYM, and (optionally, if CSW=Y) SET commands, are made part of the user profile that is in USERLIB and these values are then used to set up the task profile whenever the user initiates a new task.

Caution: Any changes to the task profile made in the current task before the PROFILE command is entered become a part of the permanent profile.

Programming Notes: This command is used when the user wants his current task profile to be used for subsequent tasks.

Examples:

1. The user wants to save task profile with no command symbol definitions.

   User: profile
   System: _

2. The user wants to save task profile including command symbols.

   User: profile csw=y
   System: _

PUNCH Command

This command schedules a job to punch an existing VSAM or VISAM data set into cards on a high-speed punch.

Operation	Operand
PUNCH	DSNAME=data set name[,] [,STARTNO=starting position][,ENDNO=ending position] [,STACK={1|2|3|EDIT}][,ERASE={Y|N}][,FORM=card form]

DSNAME
identifies the cataloged VSAM or VISAM data set to be punched.

Specified as: a fully qualified data set name.

[,]
specified, if following parameters are entered in positional notation, to maintain system compatibility.

STARTNO
specifies the byte position at which punching is to start for each data set record.

Specified as: from one to six decimal digits.

System default: punching starts 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 is punched.

Specified as: from one to six decimal digits. The value must be greater than the value of the STARTNO operand.

System default: punching continues to byte 80 or to the end of the record, whichever occurs first.

STACK
specifies 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 a FORTRAN control character or machine code (see Appendix D), but must be of the same type throughout the data set. The control character in each record is supplied by the user.

System default: 1.

ERASE
specifies that the cataloged data set is to be erased from the catalog after the punch operation is finished.

Specified as:

Y - erase.
N - save.

System default: N.

FORM
designates the punch card form to be used for this punch request.

Specified as: from one to six alphameric characters.

Note: The system does not check the specified form type; you must convey the meaning of the specified form type to the system operator.

System default: installation's standard card form is used.

Functional Description: This command results in the assignment of the request to an independent nonconversational task, to which the system assigns a BSN for possible reference by the user.

The specified data set is punched as it stands, with no code conversions. The STARTNO and ENDNO options allow selection of any contiguous field of up to 80 bytes of EBCDIC data from each record of the data set.

Input records containing an invalid control character, when the EDIT option is used, are printed in hexadecimal form on system output (SYSOUT).

If the user specifies a form number, the system includes that number in its instructions to the system operator when the card punch is readied for operation.

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.

Caution: 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 BSN to identify his task when entering the CANCEL command.

If the user wants to punch VISAM data sets, and he does not want to include the line number and keyboard/card reader character, he can specify that the data to be punched begins in column 9 of the data set (columns 1-7 are the line number; column 8 is the keyboard/card reader character). To do this, the user can specify:

```
punch dsname,startno=9
```

When the PUNCH command is used to punch a line data set, and the punched deck will subsequently be used as card-reader input by the operator to re-create that line data set, the line numbers in the original data set should not be punched out. When the operator reads the cards into the system, a line number is automatically prefixed to each record of the line data set (see Appendix A).

Example: The user wants to punch characters 24 through 56 of each EBCDIC record in 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:  BSN=0244
```

PUSH Command

This command saves the status of the interrupted program.

Operation	Operand
PUSH	[SIRTEST={N|Y}]

Note: PUSH should be issued only after an attention interruption.

SIRTEST
specifies whether the system will check for a user-defined SIR routine.

Specified as:

N - the system does not check.
Y - the system checks. If a SIR routine exists, PUSH is canceled.

System default: N.

Functional Description: PUSH saves the status of all general registers and the PSW for the active program in a system save area. If SIRTEST=Y, and if there is an active SIR routine defined by the user, the PUSH command is canceled.

Programming Note: The status of a program is automatically saved when the user issues an ATTENTION interruption. The PUSH command allows him to save the status of a program, modify the copy that he interrupted, and execute both the original and the modified copy. (See Example 2, below.)

Examples:

1. The user has interrupted his program; he wants to save the status:

   User: (presses ATTENTION key)
   System: !
   User: push

   The system saves the status of the program in a system save area.

2. The user now wants to change the copy that he just saved. Next, he wants to run the altered copy, and then run the original copy that was saved:

   User: push (from Example 1)
   Sys,User: set 5r=x'10' (alters program)
   Sys,User: go (runs altered version)
   Sys,User: go (runs original version)

QUALIFY Command

This command allows the user to reference the internal symbols within an object module without using the fully qualified name.

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, control section name, or entry point name and, optionally, a link-edited module name followed by a period and an object module name.

Functional Description: An ISD must have been requested when the original program was assembled, compiled, and, optionally, processed by the linkage editor. QUALIFY enables the user to reference, implicitly, the program's internal symbols wihtout preceding the internal symbols with the appropriate object module, control section, entry point or link-edited module name. QUALIFY loads the module if it is not currently loaded.

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: _

   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

   The system informs the user that the module has no ISD. The user must then use external symbols in his references to internal symbols in 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

   The system informs the user that he needs two levels of qualification.

   User: qualify pgml.pgmf
   System: _

REGION Command

This command is used to change regions when the user is editing a region data set (see "EDIT Command"); it can also be used to specify editing of a region data set when that information was not provided in the EDIT command.

Operation	Operand
REGION	[RNAME=region name]

RNAME
identifies an existing region or specifies the new region name to be assigned to a line or range of lines. The value of the REGSIZE operand in the user profile determines the maximum size of the region names for the data set; region names are padded on the right with blanks or truncated to fit the region name field.

Specified as: an existing region name or as a string of 1 to 244 characters.

System default: a blank region name is assumed.

Functional Description: A region data set is created with the EDIT command using the RNAME and REGSIZE operands. If the user wants to operate on another region in the same data set, he can use the REGION command to specify the new region. The system responds with an underscore if the region exists or line number 100 if the region is new to the data set. The lines of data in a region data set are prefixed by the system with the name of the region and the line number within the region. Like EDIT, the REGION command prompts with line numbers in increments of 100 (if the value of INCR is 100), until a command preceded by a break character is entered. The region name prefix is not issued when the system prompts with line numbers, but is issued when the user displays the lines with a LIST command.

Caution: The system automatically reorganizes the regions of a data set into alphabetically ascending order. A language-processing command (EDIT, PROCDEF, or PLI) must be invoked before the command is issued.

If editing commands with N1 and N2 operands are not preceded by a REGION command, the system assumes 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 he can add new lines to the region.

To create a region data set, the user enters an EDIT command with the RNAME and REGSIZE operands. The system prompts with a line number. Then, if the user wants to operate on a new region in that data set, he can enter the REGION command with the name of the new region.

If the user wants to create several region data sets in the same task, and he wants them all to have a maximum region name length of 12, he can first enter the DEFAULT command to set REGSIZE=12. He need not specify the REGSIZE operand in EDIT. All of his data sets will be created as region data sets with REGSIZE=12.

Note: Once a region data set is created, the user cannot change the maximum region name length (REGSIZE) for that data set.

To terminate processing by the REGION command, the user must issue a command preceded by a break character after the text editor line number prompt. The CLP is set to the last line entered plus the value of INCR.

Example: The user issues the EDIT command to create a new region, and then he wants to operate on another new region:

```
User:       edit dsnam,regnam,regsize=8
Sys,User:   0000100 this is data
            0000200 _region regnam2
System:     0000100
```

The system has terminated processing of the first region, REGNAM, in data set DSNAM. It has initiated processing of region REGNAM2 in the same data set.

RELEASE Command

This command deletes the data set definition established by a previously issued DDEF command. It may also be used to separate and release one or all data sets of a given concatenation (see description of DDEF in Appendix D for concatenation), or to remove JOBLIBs from the user's program library list.

Operation	Operand
RELEASE	DDNAME=data definition name[,DSNAME=data set name] [,{SCRATCH|HOLD}][,{SCRATCH|HOLD}]

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 separated 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.

SCRATCH
specifies the private volume(s) on which the affected dataset(s) resides is no longer required by the current task and may be dismounted or handed off to another task. This applies only to a volume (s) for which no other data definitions exist.

Specified as: SCRATCH

System Default: See below under HOLD command.

HOLD
specifies that the private device(s) used for the volume(s) on which the affected dataset(s) resides is to be reserved for further use by the current task. This is implied for any device(s) in use for other data definitions.

Specified As: HOLD

System Default:
for conversational task -- SCRATCH alone
for non-conversational task -- HOLD alone.

Functional Description: RELEASE deletes the information defining the data set for either a public or private data set. It also may release for other use volumes and I/O devices currently assigned to a specified private data set. If the data set is open, it is 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 JOBLIB chain, and the definition of the data set is deleted.

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 spcified, the user is prompted to enter a DSNAME.

A RELEASE command with the DSNAME parameter specified must be issued for each data set to be released from a concatenation.

When there is more than one data set in use 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. The user 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.

The LOGOFF command releases all data definitions in your task.

Examples:

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

   User: release ingo

   The system deletes the current definition of the data set.

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

   User: release tables

   The system prompts the user to enter a DSNAME.

   User: dtab1
   Sys,User: release tables,dtab2
   Sys,User: release tables,dtab3

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

   User: release overt,turn9

   The system releases TURN9 from the concatenation.

4. The user wants to release a job library with DDNAME PROGTEST.

   User: release progtest

   The system removes the JOBLIB from the JOBLIB chain.

5. The user wants to release DDNAME SAM, on a private volume. He wants to release the volume, but hold the device. He can enter either of the following:

   User: release sam,,scratch,hold

   or

   User: release sam,,hold,scratch

Operational Note: The following chart indicates volume and device disposition for the three valid combinations of SCRATCH and HOLD. It must be remembered that SCRATCH is ignored for any volume mounted for another DDEF, and HOLD is assumed for any device on which a volume is mounted for another DDEF.

	OPTIONS		DISPOSITION		TAPE
	SCRATCH	HOLD	VOLUME	DEVICE	OPTION
	-------	----	--default according to task mode--		---
(1)	SCRATCH	----	logical dismount	release(3)	rewind
(2)	-------	HOLD	retain	reserve	none
	SCRATCH	HOLD	logical dismount	reserve	rewind & unload

(1) = conversational default
(2) = non-conversational default
(3) = non-conversational reservation from SECURE is released

REMOVE Command

This command deletes previously issued dynamic statements. (See "Dynamic Statement" under "Program Control" in Section 3 of Part II and the description of the AT command earlier in this part).

Operation	Operand
REMOVE	{statement number[,...]|ALL}]

statement number
identifies a dynamic statement that is to be deleted.

Specified as: the number assigned by the system when the dynamic statement was entered.

ALL
all PCS dynamic statements are removed.

Specified as: ALL

Functional Description: REMOVE permanently cancels all dynamic statements whose numbers are specified as operands.

Caution: A REMOVE command cannot be used in a dynamic statement.

Example: The user wants to remove dynamic statements 10, 2, and 4.

User: remove 10,2,4

The system deletes the specified dynamic statements.

RET Command

This command changes the attributes that were assigned at DDEF time for a cataloged VAM data set.

Operation	Operand
RET	DSNAME=data set name,RET=retention code

DSNAME
identifies the cataloged VAM data set whose attributes are to be changed.

Specified as: a fully qualified data set name.

RET
specifies the attributes to be changed.

Specified as: P or T; C or L; U or R, where:

P - permanent storage
T - temporary storage
C - erase at CLOSE
L - erase at LOGOFF
U - unlimited access
R - read-only access

System default: one code must be specified. When T is specified, L is assumed; when P is specified, a null value is assumed for the erase option.

Functional Description: The RET command modifies the RET field in the data set descriptor (DSD), which contains information regarding the storage type, deletion characteristics, and owner access attributes of the data set that were specified in a DDEF command or DDEF macro.

Caution: At least one of the codes must be specified. Otherwise, the command is ignored.

A user may not issue RET for a data set he is sharing.

Even though the RET command has been invoked to give a user read-only access to a data set, he may still issue an ERASE command for that data set.

A data set is not erased at logoff if a RELEASE command has been issued for it and the retention code has been specified as RET=T.

Programming Notes: When the user changes the attributes of a data set to permanent storage type (P) from temporary (T), an effective null string value is given to the deletion characteristic. When the access qualification is not specified, no change to access is made.

Examples:

1. The user wants to alter the attributes of data set ETHPRG, which is temporary and unlimited and is to be erased at LOGOFF to permanent and read-only.

   User: ret ethprg,pr

   The system changes the attributes of ETHPRG to permanent and read-only.

2. The user wants to change the attributes of a permanent data set, TEST1, to be erased at CLOSE.

   User: ret test1,c

   The system changes the attributes so that the data set will be erased at CLOSE.

REVISE Command

This command specifies a line or range of lines in the current data set or region which are to be deleted and replaced.

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

N1

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

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

LAST - last line in the current region.

System default: the value of the CLP.

N2

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

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

LAST -- last line in the current region.

System default: the value of N1.

INCR

designates the increment to be used for the replacement lines.

Specified as: from one to seven decimal digits. An all-zero increment is not valid.

System default: 100.

Functional Description: REVISE first deletes the specified line or range of lines and prompts the user with line numbers for the replacement lines. When the user enters a line and presses the RETURN key, REVISE continues line-number prompting until the range specified by N1 and N2 is completed, or a command preceded by a break character is executed. When REVISE is followed by an EXCERPT command (preceded by a break character), the lines from the data set referenced by EXCERPT are inserted in the specified range. CLP is set to the last line entered plus the value of INCR. If the result exceeds the next existing line, CLP is set to the next existing line number.

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.

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

Caution: A language-processing command (EDIT, PROCDEF, or PLI) must be issued before the command is issued.

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

If the user specifies a nonexisting line in the REVISE command, or if the line specified for N1 or N2 is the last line in the data set or region, new lines are inserted into the data set with the starting line number and line-number increment specified in the REVISE command. In this case, REVISE operates the same as INSERT.

Examples:

1. The user wants to replace the existing line 300 with data lines, in increments of 10.

   User: revise 300,incr=10
   Sys,User: 0000300 first replacement line
   System: 0000310

2. The user wants to replace lines 200 through 550 with data lines, in increments of 100.

   User: revise 200,550
   System: 0000200

   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 is 600 or less). A diagnostic message is displayed and the command is terminated.

RTRN Command

This command returns control to the user in command mode. All interrupted source lists are canceled.

Operation	Operand
RTRN	

Note: There are no operands.

Functional Description: The RTRN command, entered after the user causes an attention interruption (by pressing the ATTENTION key), returns control to the user at the terminal. The user is then in command mode, and the interrupted source lists are forgotten. All SIR and AETD macro instructions are deleted.

Example: The user has interrupted a program or command string. He does not want to resume processing in that program or command string. He enters:

User: (presses ATTENTION)
System: !
User: RTRN

The system terminates processing of the interrupted program or command string and returns control to the terminal.

SECURE Command

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

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

Notes: TA and DA must be specified in keyword format. At least one of the operands must be specified.

System programmers should see System Programmer's Guide for special operands.

TA

designates the number and type of tape devices requested.

number of devices

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

System default: no tape devices are reserved.

type of device

Specified as:

7 - seven-track tape, data converter not required.
7DC - seven-track tape with data-converter feature.
9D2 - 9-track tape with 800 bpi capability
9D3 - 9-track tape with 1600 bpi capability
9D4 - 9-track tape with 6250 bpi capability

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

DA

designates the number and type of direct access devices requested.

number of devices

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

System default: no direct access devices are reserved.

type of device

Specified as:

2311 - disk
2314 - disk
3330 - 3330-1 disk
333B - 3330-11 disk

System default: the type of direct access device specified at system generation.

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. The devices that are reserved remain assigned to the task until the task logs off, or until a RELEASE command is specified with the SCRATCH option only.

Caution: 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. This is also true for tasks initiated by EXECUTE or BACK. If SECURE is not specified immediately after the LOGON command, the user's task is terminated.

Programming Note: 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 requires one 3330-11 disk drive and three tape units (at 1600 bpi), all for nine-tape. He prepares this SECURE command for insertion immediately after the LOGON command:

   ```
   secure (ta=3,9D3),da=1,333B)
   ```

2. The user's nonconversational task requires three 3330-1 disk drives and seven tape units, three of them seven-track, and the remainder nine-track (800 bpi). He prepares this SECURE command:

   ```
   secure (da=3,3330),(ta=3,7),(ta=4,9D2)
   ```

SET Command

This command changes the contents of a data location.

Operation	Operand
SET	{data location=value}[,...]

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 constant, 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 (one, two, or four bytes), floating-point (eight 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 refer to 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 cannot refer to 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 (that is, floating-point variables must be four or eight bytes; integer and logical variables must be one, two, or four 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' gives the same result; however, C'ABC' results in a diagnostic message.

Examples:

1. The user wants to set two four-byte variables with qualified internal symbols of I and K to the values of 33 and 176, respectively.

   User: set i=33,k=176

   The system sets the values.

2. The user wants to set a six-byte field to read "system."

   User: set field='system'

   The system sets the 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)

   The system sets the values.

4. The user wants to set a one-byte variable, SAM, to the binary value '10010011':

   User: set sam=b'10010011'

   The system sets the binary value.

5. The user wants to set the value of a variable (A) to 1. He wants this value entered in his user profile. He enters:

   User: set a=1
   Sys,User: profile csw=y

SHARE Command

This command allows the user to share another user's data sets.

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 which he is going to access. 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 owner of the data sets to be shared.

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.

Functional Description: 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 intial entry in the owner's catalog if "*ALL" is specified.

The user may issue the SHARE command before the owner has issued the PERMIT command to grant access to his data set or data sets. The PERMIT must be issued, however, before the user can reference or access the owner's data sets.

Cautions: The OWNERDS operand must have the same value as the DSNAME operand the owner uses 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. For 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

   The system makes the entry in the 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

   The system makes the 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 name A.A, A.B, and A.C, the user refers to them as Z.A.A, Z.A.B, and Z.A.C, respectively.

SPACE Command

The SPACE command causes the SYSOUT to be spaced the specified number of lines.

Operation	Operand
SPACE	NUMLINES=(number lines to space)

NUMLINES
number of lines to space the SYSOUT

Specified As:

1 - space 1 line (insert 1 blank line)
2 - double space (insert 2 blank lines)
3 - triple space (insert 3 blank lines)

System Default: 1 space 1 line.

Functional Description: The SPACE command module issues a 'GTWRC' macro with the appropriate carriage control character to cause the specified number of spaces (blank lines) to appear in the sysout.

Example: User wishes to separate the output from command A, from the output of the next command B.

```
User:     command A
          SPACE 3
          command B

System:   .....output from A.....

                (three blanks lines inserted
                by the SPACE command)

          .....output from B.....
```

STACK Command

This command displays all active, user-invoked module names displayed in descending order beginning with the most recent module name.

Operation	Operand
STACK	

Note: There are no operands.

Functional Description: The system displays the names of all user-invoked modules that have been interrupted and are now saved for later execution. The most recent module is displayed first. If a SIR routine is active, the user is notified; the SIR module name does not appear in the module-name display.

Example: The user has interrupted his source list. He wants to see what modules (or commands) he has interrupted:

User: stack

The system displays the active module or command names. The name at the top is the most recently interrupted program.

STET Command

See "DISABLE, ENABLE, POST, and STET Commands."

STOP Command

This command suspends execution of an object program and optionally (if LIMEN=I) 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; that is, the instruction location, expressed symbolically, at which execution is stopped.

2. Program status information (for example, 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 assembler language programs. For FORTRAN programs, it is the nearest statement number, with an increment to indicate which statement after the numbered statement has control.

Caution: STOP should appear last in a dynamic statement, because any subsequent commands are ignored, and no diagnostic is issued.

Programming Note: After an object program has been halted, the user can cause resumption of execution with the GO command.

Examples: (LIMEN has been defaulted to I)

1. The user wants to learn the status of his program when execution reaches a specified point.

User: at ftnpgm.100(4);stop

When execution reaches FTNPGM.100(4), the system replies by giving statement number assigned to above statement. The system 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 was 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

STRING Command

This command displays commands and program calls not yet executed from the current source list.

Operation	Operand
STRING	

Note: There are no operands.

The STRING command works only if it is issued immediately following an attention interruption.

Functional Description: STRING displays statements, from the current source list, that have not yet been processed.

If the user issues an attention interruption while the system is executing a PROCDEF, he receives a "PROCDEF ACTIVE" message. Then, if he issues an EXIT, or presses the carriage return (to return control to his source list), the remaining commands in the PROCDEF are executed before the displayed source list is processed. The individual commands, in the PROCDEF, are not displayed by STRING.

If the system is processing an OBEY when the STRING command is issued, the user receives an "OBEY ACTIVE" message before the source list is displayed.

Example: The user interrupts his program, and then he wants to see what commands have not been processed from the source list:

User: (presses the ATTENTION key)
System: !
User: string

The system displays unprocessed commands from current source list.

SYNONYM Command

This command renames commands or command statements, keyword operands, and PCS operands.

Operation	Operand
SYNONYM	{term=[value]}[,...]

term
designates the new name of a command, keyword, or PCS expression. This is the synonym.

Specified as: a normal or quoted string from one to eight characters.

value
specifies the value of the term that is to be used when the term is referred to. This value overrides any string value previously equated to the term. This is the old name.

Specified as: a normal or quoted string; maximum length, 244 bytes.

System default: any previously assigned synonym term is deleted.

Functional Description: The system adds, replaces, or deletes entries in the user's dictionary, according to the parameters of the SYNONYM command. When the user has assigned a value to an operand with the SYNONYM command, he can then enter a command or parameter which has a synonym value and the system uses that synonym value rather than the command or parameter entered. Synonyms may be equated to other synonyms.

The synonym and its value are only valid for the remainder of the current task, unless the user's task profile is made permanent by his issuing the PROFILE command. (See the description of the PROFILE command in Section 6 of Part II).

Caution: The user should be careful to avoid eventually equating a synonym to itself when creating a synonym chain. This creates a loop, which is broken by the system after an excessive number of synonym searches.

Programming Note: The SYNONYM command can be used to delete a synonym entry that was previously defined. The user enters the SYNONYM command and equates the operand he wants to delete to a null string by pressing the carriage return (SYNONYM A=). When a user creates a synonym, he can still refer to the command, operand, value, or expression by its original name.

Examples:

1. The user issues this command statement.

   User: synonym a=b,b=c,c=d,d=e;a

   The system calls the procedure or program named E.

   Note: The value of last SYNONYM issued overrides previous values.

2. The user executes a series of commands.

   ```
   synonym pg=pgapars
   progra pg=x,y,z
   ```

   The first SYNONYM command defines a synonym for one of the valid keywords of PROGRA. When the command PROGRA (which was previously defined by the user as a BUILTIN) is called, synonym substitution occurs, and the command is executed as:

   ```
   progra pgapars=x,y,z
   ```

3. The user creates an abbreviation for the EXECUTE command.

   synonym x=execute

   when he enters X as a command, the system invokes the EXECUTE command.

4. The user wants to nullify the synonym created in Example 3 above:

   synonym x=

TIME Command

This command establishes the time during which a task can be executed.

Operation	Operand
TIME	[MINS=minutes]

MINS
: specifies the number of minutes of execution time before the timer interrupts the task.

  Specified as: a decimal number greater than 0 and less than 451.

  System default: the value assigned at system generation.

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 is always the value 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 the time-slice has expired, no time is charged.

At the end of the time, if the task is conversational, a message is issued, and control returns to the user in command mode. If the task is nonconversational, the task is terminated abnormally.

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

The system resets the timer.

TRANSLAT Command

The TRANSLAT command is used to set the user's input and output translation tables.

Operation	Operand
TRANSLAT	TYPE,FROM,TO,USN,CP

TYPE
specifies which table is to be set, input or output.

Specified As:
OUT or O for output translation table
IN or I for input translation table

FROM
is a list of characters which are to be translated to the 'TC' character. Quotes are required as defined in rules for character strings.

Specified As: A single character or string of characters enclosed in parentheses and separated by commas. Hexadecimal values should be in the X'NN' format where NN is the hexadecimal value.

TO
is the character the 'FROM' characters are to be translated to.

Specified As: A single character -- quotes are required as defined in the rules for character strings -- or a hexadecimal value in the format X'NN' where NN is the hexadecimal value for the character.

USN
for the MTT Administrator, the number of the user for whom the command applies

Specified As: A decimal number between 0 and 128.

System Default: task owner's sysin/sysout.

CP
specifies which sysin or sysout is to be changed -- currently not supported.

Functional Description: Each sysin/sysout has a set of translate tables which users can tailor to their own terminal. All system input and output is translated by these tables. The user can achieve the same result by using SYSTRIN and SYSTROUT with the MCASTAB command. The difference between TRANSLAT and the SYSTRIN/SYSTROUT mechanism is that the TRANSLAT change is effective immediately and the SYSTRIN/SYSTROUT tables are not changed.

Example: The user wishes to change all slashes (/) to commas (,) on input:

```
User:   TRANSLAT TYPE=I,FROM=/,TO=','
User:   DISPLAY '/////'
System: ,,,,,
```

Now on output the user wants all commas to be printed as slashes:

```
User:   TRANSLAT TYPE=OUT,FROM=',',TO=/
User:   DISPLAY '/////'
System: /////
```

On output the user decides to have certain unprintable characters printed as a period (.):

```
User:   TRANSLAT TYPE=O,FROM=(X'00',X'01',X'02',X'20',X'21',X'22',
        X'23',X'24'),TO='.'
```

Now, any of the 'FROM' characters will be printed as a period (.) on the sysout.

TRAP Command (System 370 Only)

This command requests notification when execution of an object program causes certain events to occur. TRAP also designates the class of event and range of object program instruction locations in which the commands following TRAP in the dynamic statement are to be executed.

Storage Class:

Operation	Operand
TRAP	{FETCH|STORE|REF},[location[:location]]

General Register Class:

Operation	Operand
TRAP	GR,{nR,...|nR:nR}

Branch Class:

Operation	Operand
TRAP	BRANCH[,location[:location][,location[:location]]]

FETCH
Specifies that TRAP is to monitor instruction fetches within the location range specified.

STORE
Specifies that TRAP is to monitor data stores within the location range specified.

REF
Specifies that TRAP is to monitor both fetches and stores within the location range specified.

GR
Specifies that TRAP is to monitor changes in the contents of the general registers specified.

BRANCH
Specifies that TRAP is to monitor successful branches from the first location range specified into the second location range specified.

location
Specifies location or range of locations within the task's virtual memory.

Specified as: An internal or external symbol, with or without offset or subscript, or a hexadecimal address.

nR
Specifies a general register or range of general registers.

Specified as: An integer from 0 to 15 inclusive.

Functional Description: TRAP becomes effective, subject to the ranges specified in the command, at the end of the execution of an instruction when one or more of the following events has occured: 1) instruction fetch, 2) data stored, 3) change in the contents of a general register, 4) successful branch. A command statement containing a TRAP is called a dynamic statement. Only one TRAP may be included in a dynamic statement, and it must be the first command in the statement. The system assigns a number to each dynamic statement. This number may be referenced by the REMOVE command. Only one TRAP in each of the three classes may be outstanding at any point in time. The system will automatically remove a TRAP statement if an attempt is made to issue more than one TRAP in one class.

Coincident events may occur and are processed in the following order:

1. Storage
2. General Register
3. Branch

Unprocessed coincident events are lost if program execution is resumed by a CALL or BRANCH statement.

When a TRAP command is executed, a standard output (including the instruction location where the command became effective, program status information, and the dynamic statement number) is presented to the user. If LIMEN is not set to I, only the dynamic statement number is displayed. 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. Execution of a TRAP command is disabled during execution of privileged system programs. Also, SVC instructions and instructions that cause program interrupts to occur will not cause a TRAP command to be executed.

The counter, referred to by the special character %, is assigned to a dynamic statement and is incremented by one when the TRAP statement is executed. The counter is incremented even when the dynamic statement is conditional. The counter may be used as an operand in other PCS commands within the statement. The TRAP command alone will interrupt, but not stop, program execution.

Programming Notes: If TRAP specifies FORTRAN statement numbers as locations, the numbers must designate executable FORTRAN statements and not format statements. The determination of whether a successful branch falls within a specified location range in a TRAP branch statement is performed interpretively by PCS for all successful branches. Therefore, this statement should be used with care.

Example: The user wants to be informed when his program alters the contents of general register 12.

To accomplish this,

User: trap gr,12r
System: 0001

Execution of the program begins. When general register 12 is altered, the user is notified. For example, the system prints out the following line (assuming LIMIN=I):

System: TRAP SWTC.(X'2E') PSW 2 0 F 00335032 0001

In this statement
SWTC.(X'2E') is the location of the instruction that altered general registet 12. n that altered general register 12.

PSW 2 0 F 00335032 is the program status.

0001 is the dynamic statement number assigned by the system.

Note: If LIMEN had not been set to I, only 0001 would have been printed by the system.

TV (Tape to VAM) Command

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

Operation	Operand
TV	DSNAME1=tape data set name [,DSNAME2=vam data set name],*, [,OVERLAY= Y|N] [,RETAIN= Y|N] [,FROMID=user identification] [,TOID=user identification]

* This position has no meaning for the TV command but is present in the BPKDS for ease of coding.

DSNAME1
identifies, in the absence of a previously defined DDEF command with the DDNAME of DDTVIN, an 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 by an asterisk, this data name must be preceded by an asterisk here.

Where a previously defined DDEF command with a DDNAME of DDTVIN exists, it identifies one of all data set names that are to be restored to direct access storage.

Specified as: the fully qualified data set that will be located on the volume(s) specified by the DDEF command with the DDNAME of DDTVIN, or

*ALL; all data sets on the volume(s) specified by the DDEF command with DDNAME of DDTVIN will be processed.

DSNAME2
specifies the name under which the data set will be restored. 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, or *DSNAME1, the data set name that was retained on tape is to be used for the name of the data set that is to be restored on direct access storage.

System default: a name will be generated by the system in the form:

$D.Dnnnn.dsname1

and used as the VAM data set name. Truncation, if required, will be performed from left to right to allow the insertion of $D.Dnnnn. The $D qualifier will allow the user to reference all such data sets created by TV by partially qualified data set name.

OVERLAY
specifies that the output data set will be overlayed if it already exists and the data set attributes (DSORG, RECFM, LRECL, RKP, KEYLEN) for the two copies are the same.

Specified as: Y - overlay to be made.
N - overlay not to be made.

System default: N - no overlay to be made.

RETAIN
specifies that the change and reference dates of the input data set are to be retained with the output data set.

Specified as: Y - input dates are to be retained.
N - current dates are to be retained.

System default: the current dates will be retained with the data set copy.

FROMID
specifies the user identification prefixed to the name of the input data set.

Specified as: One-to-eight alphameric characters, or *ALL; all userids retained on the tape are to be processed.

System default: the user identification associated with current task is assumed.

TOID
specifies the user identification to be associated with the output data set.

Specified as: For a user, this must be the user identification associated with the current task.

For the system manager, any user identification currently joined to the system.

For a system administrator, any user identification currently joined to the system by him.

*FROMID; the original user identification saved on tape is to be used. It must be user identification currently joined to the system and one which the user has authority to specify.

Functional Description: For each successfully copied data set, the user is 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 results in a diagnostic message and cancellation of the command.

Generation indexes will be created as necessary for generation data groups not previously defined in the system.

The DDEF command with DDNAME of DDTVIN must specify PS as the data set organization (DSORG), nine-track tape as the residence volume (UNIT) and unlabeled tape (LABEL).

An entire tape may be restored to a private VAM volume through the use of a DDEF command with DDNAME or DDTVOUT. This DDEF must specify VAM data set organization (DSORG) and identify the device type (UNIT) and volumes (VOLUME) required to contain the tape data sets. The DDEF will be used only when a DDEF with DDNAME of DDTVIN is present and DSNAME1 is specified as *ALL.

Note: When all data sets (DSNAME1=*ALL) on a tape volume are to be processed, DSNAME2 must be specified as *DSNAME1 or be defaulted.

Examples:

1. The user wants to restore data set ABC onto a private VAM volume (MYVOL1) as XYZ.

   User: ddef dum1,vp,xyz,unit=(da,2314),volume=(,myvol1)
   tv abc,xyz

   System: (copies ABC onto private volume MYVOL1 as XYZ)

2. The user wants to locate data set SOURCE.MYDS on volume MR9024 and restore it to public storage as SOURCE.MYDS. The data set name LOG is used to fill the DDEF requirements for a dsname value. It may be any data set name that will allow DDTVIN to be defined.

   User: ddef ddtvin,ps,log,unit=(ta,9),Volume=(,MR9024),-
   LABEL=(,nl);tv source.myds,*DSNAME1,overlay=y

   System: (locates SOURCE.MYDS and restores it to public storage)

3. The user wants to restore all data sets from volume MR9024.

   User: ddef ddtvin,ps,zip,unit=(ta,9),volume=(MR9024),-
   LABEL=(,nl);tv *all,*DSNAME1,overlay=y

   System: (restores all data sets from volume MR9024)

UNLOAD Command

This command removes a module and all other modules tc which 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 (see below).

Functional Description: The UNLCAD 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 referred to by one of the following commands is unloaded: PLI, ASM, LNK, FTN, LCAD, UNLOAD, CALL with a specified module name, or an implicit call.

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 re-issue the UNLOAD command later, if desired.

Caution: When a module is unloaded, all PCS AT statements are removed for that module (reloading the module does not replace these AT statements). The user is notified of the AT statements that are removed. However, the AT number counter is not reset to 0 unless all AT statements are removed from all modules.

Programming Note: 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: The user wants to unload a module named ABC.

User: unload abc

The system unloads ABC and all modules to which ABC implicitly or explicitly refers.

UPDATE Command

This text-editing command adds or inserts the data lines entered at the terminal into the current data set or region.

Operation	Operand
UPDATE	

Note: There are no operands.

Functional Description: UPDATE unlocks the keyboard to prompt the user to enter a line number, a blank or tab, and data. The lines of data entered by the user are inserted in the current region at the specified line number. If the user specifies a line number that already exists in the region, the new line overlays the old line.

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

Cautions: When issuing insertion lines, one space or tab must be entered between the line number and the text of the line. Excessive tabs or spaces are treated as text.

A language-processing command (EDIT, PROCDEF, or PLI) must be issued before the command is issued.

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: (unlocks the keyboard)
User: 250 data1
System: (inserts line 250 between 200 and 300 and unlocks the keyboard)
User: 650 text
System: (inserts line 650 between 600 and 700 and unlocks the keyboard)
User: 500 more data
System; (replaces old line 500 with new line and unlocks the keyboard)

User: _insert 1100
System: (terminates execution of UPDATE, positions the CLP to line 1100, and prompts the user to enter line 1100 which does not yet exist)

USAGE Command

This command presents to the user the statistics accumulated in the system that relate to his use of system resources.

Operation	Operand
USAGE	

Note: There are no operands.

Manager's and Administrator's Guide and System Programmer's Guide list special operands for managers, administrators, and system programmers.

Functional Description: The accounting statistics for the specified user identification, which include, the user's ration (that is, the maximum amount of each resource allowed for the user), the accumulative statistics in the user table, and the usage statistics for the current task are tallied and presented to the user. Conversationally, the data set is presented at the terminal; nonconversationally, in the SYSCUT data set.

The user's accounting statistics are displayed always. The accumulative statistics cannot be reset to zero without displaying all of the user's accounting data. Furthermore, the display of his statistics occurs before the accumulative data is reset to zero, although subsequent entry of the USAGE command displays the statistics with their new values.

Three types of statistics are presented: accumulative statistics reflect total usage of resources from the time the user is joined to the system to the present, current statistics reflect usage during the present task, and the ration which reflects the maximum amount of a resource allowed for the user. The statistics displayed are summarized in Appendix H.

The information is presented in the following format:

```
/TEMP STOR=ration;current;accum
/PERM STOR=ration;current;accum
/DA DEV=ration;current;accum
/MAG TAP=ration;current;accum
/PRINTERS=ration;current;accum
/RD-PUN=ration;current;accum
/TSS TASKS=ration;current
/BULKIN=accum
/BULKOUT=accum
/CPU TIME=ration;current;accum
/CONN TIME=ration;current;accum
```

Statistics with zero value are not presented.

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], [,ERASEDS1= Y|N],*, [,RETAIN= Y|N] [,FROMID=user identification] [,TOID=user identification] [,CATDS2= Y|N]

* This position has no meaning for the VT command but is present in the BPKDS for ease of coding.

DSNAME1
identifies the cataloged VAM data set to be written on magnetic tape.

Specified as: a fully qualified data set name.

DSNAME2
specifies the name to be assigned to the magnetic-tape copy of the data set.

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, or

*DSNAME1, the copy is to retain the same name as the original data set.

System default: for the first VT command, the data set name given in the preceding DDEF command, with the DDNAME of DDVTOUT is assumed. For subsequent VT commands, the tape data set name will be modified to the form of

$T.Tnnnn.dsname1

and used as the tape data set name. Truncation of the given name to allow the insertion of $T.Tnnnn (where nnnn is a unique number assigned by the system to assure data set uniqueness) will be from left to right. The qualifier $T will allow the user to reference all such data sets created by VT by partially qualified data set name. If this operand is specified as *DSNAME1 the tape data set name will be the same as the vam data set name and no cataloging will be performed.

ERASEDS1
specifies that the VAM data set is to be erased after the data set

Specified as: Y - erase after copy
N - no erase after copy

System default: N - no erasure will be made.

RETAIN
specifies that the change and reference dates of the input data set are to be retained with the output data set.

Specified as: Y - input dates are to be retained.
N - current dates are to be retained.

System default: the current dates will be retained with the data set copy.

FROMID
specifies the user identification prefixed to the name of the input data set.

Specified as: For a user, this must be the user identification associated with the current task.

for the system manager, any user identification currently joined to the system.

For a system administrator any user identification currently joined to the system by him.

System default: the user identification associated with the current task is assumed.

TOID
specifies the user identification to be associated with the data set on tape.

Specified as: One-to-eight alphameric characters, or

*FROMID; the FROMID is to be retained on tape as the TOID.

System default: The user identification, associated with the current task is assumed.

CATDS2
specifies whether or not the data set on tape is to be cataloged.

Specified as: Y - the tape data set is to be cataloged.
N - the tape data set is not to be cataloged.

Functional Description: The VT command can be used to copy data sets serially on tape without issuing a new DDEF command each time. Once the user has identified the output data set by a DDEF command with DDNAME of DDVTOUT, VT accepts each new request, updates the required control information, and copies 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.

The DSNAME2 and TOID values will be retained as part of the tape data set. It may be used by the TV command when the data set is restored to direct access storage.

Although the user may specify that the output data set be cataloged (CATDS2=Y), DSNAME2 will not be cataloged where *DSNAME1 is specified, the name is preceded by an asterisk, the name is already cataloged or the TOID is not for a userid currently joined to the system.

DSNAME1 will be erased (ERASEDS1=Y) only if the data set is copied successfully.

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

Note: Before the initial VT command in a task, a DDEF command must be issued for the tape data set with DDNAME of DDVTOUT.

For each successfully copied data set, the user receives 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 results 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 operand. In addition, when the TV command will be used to process the entire tape volume (identified to TV by a DDEF with DDNAME of DDTVIN), the DDVTOUT DDEF must specify an unlabelled tape in the LABEL operand.

Example:

1. The user wants to write his public 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 ABC)

2. Now the user wants to write his public VAM data set, WN3, on to the same private tape volume.

   User: vt dsname1=wn3

   System: (copies WN3 on tape with data set name $T.T0003.WN3)

3. The user wants to save his VAM dataset DATA3 on a private tape as DATA3.

   User: vt data3,*DSNAME1

   System: VSN (vsn) FSQ (fsq) data3 saved on tape.

VV (VAM to VAM) Command

This command copies a VAM data set in direct-access storage.

Operation	Operant
VV	DSNAME1=current data set name [,DSNAME2=new data set name] [,ERASEDS1= Y|N][,OVERLAY= Y|N] [,RETAIN= Y|N] [,FROMID=user identification] [,TOID=user identification]

DSNAME1
identifies the cataloged VAM data set to be copied.

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

where nnnn is a unique number assigned to assure uniqueness of data set names. The qualifier $D will allow the user to reference all such data sets created by VV by partially qualified data set name.

ERASEDS1
specifies that the input data set is to be erased after the data set has been copied.

FROMID
specifies the user identification prefixed to the name of the input data set.

Specified as: For a user, this must be the same as the user identification associated with the current task.

For the system manager, any user identification currently joined to the system.

For a system administrator, any user identification currently joined to the system by him.

System default: the user identification associated with the current task is assumed.

Specified as: Y - erase after copy
N - no erase after copy

System default: N - no erasure will be made.

OVERLAY
specifies that the output data set will be overlayed if it already exists and the data set attributes (DSORG, RECFM, LRECL, KEYLEN, RKP) for the two copies are the same.

Specified as: Y - overlay to be made
N - overlay not to be made

System default: N - no overlay to be made.

TOID
specifies the user identification to be associated with the output data set.

*FROMID; the FROMID is to be used as the TOID.

System default: the current task userid is assumed.

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.

DSNAME1 will be erased (ERASEDS1=Y) only when the data set is copied successfully.

Generation indexes will be created as necessary for generation data groups not previously defined in the system.

Examples:

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

User: vv xyz

System: (copies XYZ with the name $D.D0003.XYZ)

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

User: ddef dummy,vi,abc,unit=(da,2311),volume=(,myvol1)

vv gh2,abc

System: (copies GH2 onto MYVCL1 with name ABC)

WT Command

This command writes an existing VSAM or VISAM 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=starting position][,ENDNO=ending position] [,PRTSP={EDIT|1|2|3} [,HEADER=H][,LINES=lines per page][,PAGE=P] [,ERASE={Y|N}]

DSNAME
identifies the cataloged VSAM or VISAM data set to be written on tape in print format.

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.

System default: a previously labeled scratch tape is used.

Caution: DSNAME2 must refer to a previously labeled tape data set.

VOLUME
specifies the volume identification number of the output tape.

Specified as: from one to six alphameric characters.

System default: scratch tape is used.

FACTOR
designates the blocking factor for records of the output tape.

Specified as: from one to three decimal digits. The maximum blocking factor is 246.

System default: 30.

STARTNO
specifies, for each record, the byte position at which writing onto the tape is to start.

Specified as: from one to six decimal digits.

System default: writing starts with the first byte of each record.

ENDNO
specifies, for each record, the byte position at which writing onto the tape is to stop. This end byte is written.

Specified as: from one to six decimal digits. The value must be greater than the value of the STARTNO operand.

System default: writing continues 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 is controlled by a character in the first byte position of each data set logical record. The control character may be a FORTRAN control character or machine code (see Appendix D), but must be of the same type throughout the data set. The control character in each record is supplied by the user.

1 - one space between lines.

2 - two spaces between lines.

3 - three spaces between lines.

System default: 1.

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, is used as the header.

Specified as: H

System default: no header is printed.

LINES
designates the number of lines to be printed on a page.

Specified as: from one to four decimal digits (maximum 9999).

System default: 54 lines are printed on each page.

PAGE
specifies that pages are to be numbered.

Specified as: P

System default: no pages are numbered.

ERASE
specifies that the cataloged data set is to be erased from the catalog after the tape operation is finished.

Specified as:

Y - erase.
N - save.

System default: N.

Functional Description: WT results in the creation of an independent nonconversational task, to which the system assigns a BSN 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 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 are 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.

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: 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 BSN 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 offline 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 isto be erased after it is written on tape. The tape data set, TAPEDS1, will be system blocked.

User: wt rt.winder,tapeds1,,,20,130,2,h,60,p,erase

The system accepts the task and assigns a BSN.

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 when invoked by LOGON.

Functional Description: After the task initialization process is completed, LOGON calls ZLOGON. 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 function accomplished 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. If a program is to be executed with ZLOGON during the initialization process, it must reside in USERLIB. 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
```

   Note: Program PGMA must be stored in the user's USERLIB to be executed. The user can also run program PGMA during his task by issuing ZLOGON.

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

```
synonym zlogon=cb
```

   Note: During the LOGON process, the CB command is issued before the user gets control. CB is ignored during initiation of a non-conversational task.

APPENDIX A: BULK INPUT FROM MAGNETIC TAPE

This appendix describes how the user enters bulk input from magnetic tape.

The way described here is the only direct means of reading a data set from tape and then converting it to VAM organization, writing it onto public storage, and cataloging it. 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 that are defined later in this appendix. The data set that will be stored and cataloged has a different organization from the input data set residing on the tape 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 refer to any other cataloged data set belonging to him.

INFORMATION NEEDED BY THE SYSTEM OPERATOR

The user must send the following information with his tape to the system operator for every data set that is to be read and cataloged.

The system operator uses the information to enter an RT command that causes execution of a system-provided task to handle the tape input. Tne SYSOUT listing of that task, which is returned to the user, may contain messages.

1. Identification of the user to whom the data set belongs. This identification is specified as from three to eight alphameric characters. The first character must be alphabetic.

2. Volume identification of the tape. This identification must be specified as from one to six alphameric characters.

3. Type of tape, for example, 7 (seven-track tape), 7DC (seven-track tape with data converter feature), or 9 (nine-track tape). If the user does not specify a type of tape, the tape type specified at system generation is assumed.

   When the user wants to submit a data set on seven-track tape (with or without the data converter feature), he must first consider tape characteristics. If characteristics such as density and parity match the standards set by the installation, specify the type of tape as shown above. However, if characteristics are different, the user must issue a DDEF command for the data set, specifying the tape characteristics; issue a CATALOG command to catalog the data set; and tell the operator that the tape has been cataloged. (See Item 5.)

4. Name of the input data set, specified as a fully qualified data set name.

5. CTLG, which indicates that the data set is cataloged.

6. Name under which the data set is to be cataloged, specified as a fully qualified data set name.

7. LINE, if the user wants lines to be numbered. If this option is specified, a VISAM data set that has variable-format records is created. Otherwise, a VSAM data set without line numbering is created.

8. What action is to be taken if an uncorrectable read error occurs. One of the following options can be specified: ACCEPT (error record is accepted), SKIP (error record is skipped), END (read operation is terminated). If the user does not specify an option, END is assumed.

The system reads the input data set, converts it to VAM organization, stores it on public storage, and catalogs 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 reads 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 generates line numbers in increments of 100. The maximum number of logical records permitted 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 makes any character adjustments required for data validity.

TAPE FORMAT REQUIREMENTS

The magnetic tape must have the standard TSS label or a standard ASCII label. (This standard refers to American National Standard for Information Interchange ANSI X3.4-1968. The abbreviation ASCII is used throughout this book.) Physical records must be fixed length and no longer than 32,767 bytes.

APPENDIX B: BULK INPUT FROM CARD DECKS

This appendix describes how the user enters bulk input from cards.

The user submits his data sets on punched cards to the system operator, who enters them into the system via a high-speed card reader. A system-provided task that handles the card input is executed. A SYSOUT listing of that task is produced. That listing may contain messages. 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 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 in exactly the format used to enter commands from a terminal (see Part I under "Command Format and Notation"). The first card must be a LOGON command; the last, LOGOFF. Only the LOGON and LOGOFF commands must begin in column 3. Any command that is preceded by a break character (normally the underscore) must begin with the break character in column 1 if it is to be recognized.

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 shown in Figure 3.

The SYSIN data set may include data that is to be read by the user's object program during execution. If so, the data to be read must appear immediately after the command that starts execution of the user's program. (Card data may also start in column 1.) Also, for FORTRAN data, the end-of-data card, starting in column 1, must follow the last data card, as in Figure 4.

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 continues to reside in storage until it is specifically erased. Unlike the nonconversational SYSIN data set, it is not executed upon being read.

The format of a data-card data set is shown in Figure 5. The first card of the data set must be a data descriptor card; the last must be the %ENDDS card. The information that is punched into each card must start in column 3.

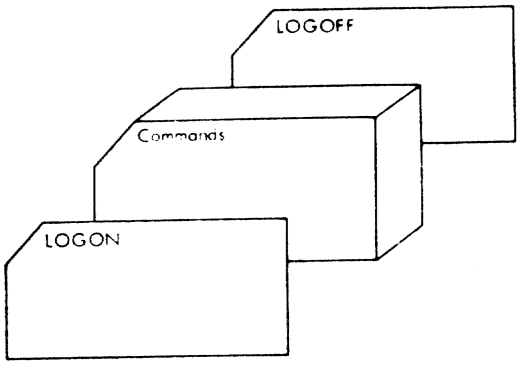


Figure 3. Card deck for a non-conversational task

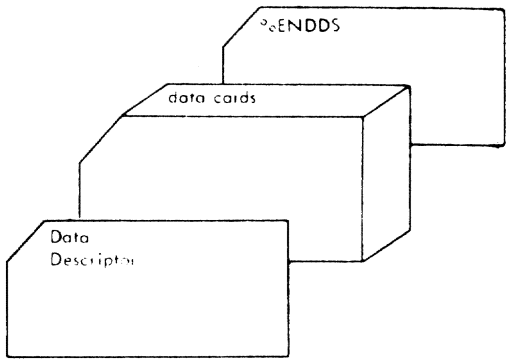


Figure 5. An example of the data-card data set

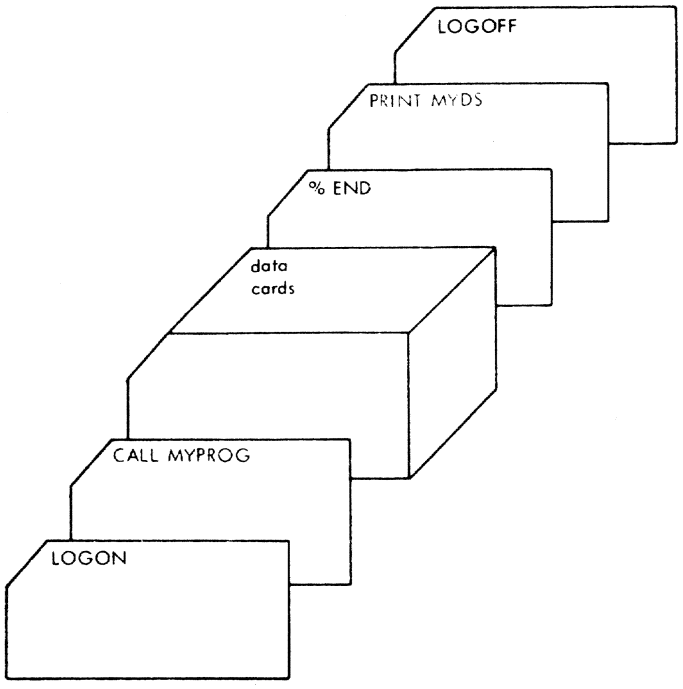


Figure 4. An example of a SYSIN data set, showing input data cards and the end-of-data card

Data Descriptor Card

The information that is given on the data descriptor card is used by the system to create a data set. The format of the information on the card is as follows:

Operands
DATASET,user identification,dsname[,format] [,starting number][,ending number] [,{LINE|FTN|COMP|CARD}] [,error][,REPLACE]

DATASET
indicates that data descriptor information follows. This operand must begin in column 3.

Specified as: DATASET

user identification
identifies the user to the system.

Specified as: the user's identification assigned to him when he was joined to the system.

dsname
is the name under which the new data set is to be cataloged.

Specified as: a fully qualified data set name.

format
designates the class of card punching to be used.

Specified as:

EBCDIC -- extended binary coded decimal interchange code.
BCD -- binary coded decimal.

System default: EBCDIC.

starting number
is the first column to be read when creating the data set records.

Specified as: a decimal number from 1 to 80.

System default: column 1.

ending number
is the last column to be read when creating the data set records.

Specified as: decimal number from 1 to 80.

System Default: column 80.

LINE
indicates that line numbering is requested. Each record in the data set is prefixed by a seven-character line number of the form xxxxx00 and by a byte of binary zeros that is reserved for system use. The resulting data set is a line data set with variable format records.

Specified as: LINE

System default: no line numbering, VSAM.

FTN
indicates that a data set with the same characteristics as defined for LINE, above, be created. In addition, however, FTN specifies that the input cards are FORTRAN source and that the user wants them converted to contain keyboard continuation conventions. The

resultant data set can be updated from a terminal without any special consideration for multicard statements. Trailing blanks are stripped from statements that are not continued.

COMP
indicates that a data set with the same characteristics as defined under LINE be created except that trailing blanks are stripped from all input records. The data set is compressed at the point as specified by the END byte. Accordingly, for example, compression could take place in an assembler source disregarding the sequence number.

CARD
indicates that a VSAM fixed-length data set be created (no line numbering). The record length is the difference between the starting number and the ending number + 1. This is the system default.

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

REPLACE
indicates whether an existing data set, with the same name as that specified on the DATASET card, will be erased when this data set is cataloged. When REPLACE is specified on the DATASET card and a data set with the same name exists in the user's catalog, the old data set is erased, and the new data set is cataloged. If, however, the existing data set is on private storage, or if REPLACE is not specified, a diagnostic is issued, and the DATASET operation is canceled.

Specified as: REPLACE

System default: The old data set is not erased; the DATASET operation is canceled.

Functional Description: The operator initiates card reading; and the system reads the input data set, converts it to VAM organization, puts it in public storage, and catalogs it in the user's catalog under the data set name provided. The stored data set has either a VSAM or a VISAM organization, depending on whether the LINE option was selected. If line numbering was requested, the system generates line numbers in increments of 100. The maximum number of lines permitted in such a data set is 100,000. When line numbering is requested, the new data set record length must not exceed 120 bytes.

Caution: You cannot create a VPAM data set with the DATASET card.

%ENDDS Card

This card, with %ENDDS starting in column 3, marks the end of a data set that is to be cataloged. (See Figure 5.)

APPENDIX C: PROTOTYPE PROFILE

The prototype profile, which is initially used to form the user profile for each user, contains command system defaults, input and output translation characters, and a table of miscellaneous control characters. This appendix covers the following topics:

- The table of system defaults
- The basics of translation, including the translation tables and the corresponding function codes for input translation and for output translation
- The character switch table

TABLE OF SYSTEM DEFAULTS

Table 19 contains default values for command operands and implicit operands. (Implicit operands are not specified with a command, but they may affect the operation of a command or of the system.) This table shows the initial values for these defaults. Each value can be changed with the DEFAULT command.

Table 19. Command system defaults

Table 19. Command system defaults (part 1 of 4)

Operand Name	Default Value	Purpose or Command that Uses
ACC		CATALOG
ACCESS		PERMIT
ACTION	O	CATALOG
ALIAS		POD?
ALPHABET	1	C,CA,CB,K,KA,KB
ASMLIST	Y	ASM
BASE	100	EDIT, REGION
BCD	N	FTN
BREVITY	T	Message length filter
BSN		CANCEL
CHAR	C	CORRECT, LIST
CLEANUP	Y	EXIT
CLP		MCAST
CONF		MODIFY
CONPRMPT	Y	UPDATE, text editor data input routine
CONREC	N	UPDATE, text editor data input routine
CONT		MCAST
COPYBASE		CDS
COPYINCR		CDS
CORMARK	*$@%#	CORRECT
CP		MCAST
CRLIST	N	ASM, FTN
CSW	N	PROFILE

Table 19. Command system defaults (part 2 of 4)

Operand Name	Default Value	Purpose or Command that Uses
DATA		POD?
DBASE		DATA
DCB		DDEF, FILEDEF
DDNAME		CLOSE, DDEF, FILEDEF , JOBLIBS, RELEASE
DEPROMPT	Y	DELETE, ERASE
DEVICE		EVV
DIAGREG	N	ABEND
DINCR		DATA
DISP		DDEF, FILEDEF
DSNAME		BACK, CATALOG, CDD, CLOSE, DATA, DDEF, DELETE, EDIT, ERASE, EXCERPT, EXECUTE, FILEDEF, LINE, PERMIT, PRINT , PUNCH, RELEASE, RET, SHARE, WT
DSNAME1		CDS, TV, VI, VV
DSNAME2		CDS, TV, VI, VV, WT
DSORG		DDEF, FILEDEF
ENDNO		PRINT, PUNCH, WT
EOB		MCAST
ERASE	N	CATALOG, CDS, PRINT, PUNCH, WT
ERROROPT	END	PRINT, PUNCH, WT
EXPLICIT		PLI
EXTNAME		BUILTIN
FACTOR		WT
FORM		PRINT, PUNCH
FROMDEV		DMPRST
FRVOLID		DMPRST
FTN		MODIFY
GDG		CATALOG
GNO		CATALOG
HEADER		PRINT, WT
HEXSW	x%	CONTEXT, DATALINE service routine, UPDATE
INCR	100	EDIT, INSERT, NUMBER, REGION, REVISE
INSERTn		PRMPT
INSTLOC		BRANCH
INTRAN		MCASTAB
ISD	Y	ASM, FTN, LNK
ISDLIST	N	ASM
JOBLIB		DDNAME?
KC		MCAST
KEYLEN		DDEF, FILEDEF MODIFY
LABEL		DDEF, DMPRST, FILEDEF
LGH		LL
LIB		LNK
LIMEN	W	Message severity filter
LINCR	(100,100)	ASM, FTN, LNK
LINE		LINE?, RT
LINENO	Y	DATA, MODIFY, text editor data input routine
LINES	54	PRINT, WT
LISTDS	Y	ASM, FTN, LNK
LOC		RUN
LPCXPRSS		ASM, FTN, LNK
LRECL		DDEF, FILEDEF MODIFY

Table 19. Command system defaults (part 3 of 4)

Operand Name	Default Value	Purpose or Command that Uses
MACRO		FILEDEF
MACRODS		PLI
MACROLIB		ASM
MAP		PLI
MERGEDS		PLI
MERGELST		PLI
MINS		TIME
MMAP	N	FTN
MNAME		QUALIFY
MODREP		ASM, FTN, LNK
MODULE		POD?
MSGID		PRMPT
N1		CONTEXT, CORRECT, EXCERPT, EXCISE, INSERT, LIST, LOCATE, NUMBER, REVISE
N2		CONTEXT, CORRECT, EXCERPT, EXCISE, LIST, LOCATE, NUMBER, REVISE
NAME		ASM, BUILTIN, CALL, COBOL, FTN, FTNH, HASM, LNK, LOAD, PLI, PLIOPT, PROCDEF, UNLOAD
NAMES		DSS?, PC?
NBASE		NUMBER
NEWNAME		CATALOG
NEWPASWD		CHGPASS
NEWVLID		DMPRST
NUMLINES	1	SPACE
OBLIST	N	FTN
DCERASE		ODC
ODCPLI		ODC
OPTION		DDEF, FILEDEF
OPTION1		EXHIBIT
OSDDN		FILEDEF, FILEREL
OSKEYLE		FILEDEF
OSOPTS		COBOL, FTNH, HASM, PLIOPT
OUTRAN		MCASTAB
OWNERDS	*ALL	SHARE
PADCHAR		PLI
PAGE		PRINT, WT
PLCOPT		PLI
PLIOPT		PLI
PLIPACK		PLI
PMDLIST	N	ASM, FTN, LNK
PODNAME	USERLIB	POD?
PREXPAND		Controls procedure expansion error analysis
PROCNAME		KEYWORD
PROLIB		BUILTIN, PROCDEF
PROTECT		DDEF
PRTSP	1	PRINT, WT
PUBLIC	N	FTN
READ	N	BLIP
RCC		MCAST
RECFM		DDEF, FILEDEF MODIFY
REGSIZE	0	EDIT
REJMSG		PLI
REPLACE		CDS
RESET	N	LL
RET		DDEF, FILEDEF RET
RKP		DDEF, FILEDEF MODIFY
RNAME		EDIT,EXCERPT,REGION
RS		MCAST
RSVP		Controls responses in GATE
RTYPE		DATA
RUNMODE		DMPRST

Table 19. Command system defaults (part 4 of 4)

Operand Name	Default Value	Purpose or Command that Uses
SCOL	0	CORRECT
SETNAME		MODIFY
SHARED		ERASE
SIRTEST		EXIT, PUSH
SLIST	Y	FTN
SOURCEDS		COBOL, FTNH, HASM, PLI, PLIOPT
SPACE		DDEF, FILEDEF
SSM		MCAST
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
SYMLIST	N	ASM
SYSIN	K	Controls GATE's access to SYSIN data set
SYSINX	G	Controls command analyzer's access to GATE
TIME		BLIP
TODEV		DMPRST
TOVOLID		DMPRST
TRANTAB	N	Text editor
TRP		MCAST
TRUNCATE	N	LL
TYPE		CLOSE, EXHIBIT
UNIT		DDEF, FILEDEF
UPDTXFER		PLI
USERID		PERMIT, SHARE
USM		MCAST
VERID		ASM, FTN, LNK
VOLUME		DDEF, EVV, FILEDEF, WT
WRITCHK		DMPRST
XFERDS		PLI

BASICS OF TRANSLATION

Each character translation table (see Table 20 and Table 21) is made up of two parts: the first part has 256 translation entries, and the second part has 256 corresponding entries that are the function codes assigned to the translation entries.

The translation entries begin at byte 0 (X'00') and continue through byte 255 (X'FF'). The function codes begin at byte 256 (X'100) and continue through byte 511 (X'1FF'). There are two sets of function codes, one for input and one for output.

The meanings of the input function codes that are shown in Table 20 are as follows:

Code 00 - Translate only
when this code is specified for a character, the system picks up the internal code of that character.

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.

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 table of miscellaneous control characters. Any characters beyond the EOB character are ignored.

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 #.

Code 10 - Terminal null
if this code is assigned to the last character before the end-of-block, the system ignores the character. The system-supplied value is a new line.

Code 14 - Null
any character to which this code is assigned is ignored as input to the system.

Code 18 - Escape
any character to which this code is assigned becomes a one-character escape. The character immediately following is always treated as data.

The meanings of the output function codes that are shown in Table 21 are as follows:

Code 00 -- Translate only
all characters assigned this function code are translated using the corresponding values in the first half of the output translation table.

Code 04 -- Restore
when a bypass code is in effect for output data, the restore code restores printing of the output data to a 1050 Data Communicatcns System.

Code 08 -- Bypass
any output data, preceded by a bypass code, is not printed at a 1050 terminal. A restore code restores output printing.

Code 0C -- Prefix
a character with this function code causes the printer ribbon to be shifted. This function requires special features on the 1052 Printer-Keyboard and the 2741 Communications Terminal.

Code 14 -- Tab
this code causes a tab to be generated at the terminal. The output continues at the next tab position. (Line length control is terminated when a tab character is recognized.)

Code 18 -- New Line
this code generates a carriage return and a line feed. The function is not recognized on the teletype terminal.

Code 1C -- Backspace
this code generates a backspace. On the teletype terminal there is no physical backspace; a left arrow is printed to indicate that a backspace has occurred.

Code 20 -- Delete Character
a character assigned this code is deleted from the output stream; initially, all unprintable characters have this code.

Code 28 -- Line Break Character
when a line of output data is too long for the terminal, this character is used to indicate where the line may be broken.

Code 30 -- Line Feed
this code causes the terminal paper to be moved up one space.

The procedure of translation is as follows:

1. A character is entered into the system, either from a terminal or from an output data set. For example, a user enters a capital letter A from the terminal.

2. The system uses the hexadecimal position of A, which is C1, as an index into the translation table. (The translation entries and the corresponding functional codes for input are shown in Table 20. These are the values that are provided intially by the system. The values can be changed.)

3. There is a code in the location in the table that corresponds to the character (really, it corresponds to the EBCDIC representation of the character). The value in Table 20 for the letter A is X'C1'.

4. The system looks at the corresponding function code position to see if there is an entry. For example, the system looks at decimal location 449 (see the column labeled "Function Code"; the hexadecimal byte is X'1C1' which is found by adding X'C1' to X'FF') for the function code for A.

5. If a function code is found, the function is performed. If no function code is found, the character is translated only. For example, the function code for A is 00 for input, as shown in Table 20 (translate only), and 00 for output, as shown in Table 21 (translate only).

The user can change his translation tables. This procedure is explained in the descriptions of the MCASTAB and SET commands in Part III.

Table 20. Prototype input character translation table (part 1 of 6)

Translation Entry			Function Code	
Byte (Decimal)	Code (Hexadecimal)	Character	Byte (Decimal)	Input Code (Hexadecimal)
0	0		256	00
1	1		257	00
2	2		258	00
3	3		259	00
4	4	PF	260	00
5	5	HT	261	00
6	6	LC	262	00
7	7	DEL	263	00
8	8		264	00
9	9		265	00
10	A		266	00
11	B		267	00
12	C		268	00

Table 20. Prototype input character translation table (part 2 of 6)

Translation Entry			Function Code	
Byte (Decimal)	Code (Hexa-decimal)	Character	Byte (Decimal)	Input Code (Hexadecimal)
13	D		269	00
14	E		270	00
15	F		271	00
16	10		272	00
17	11		273	00
18	12		274	00
19	13		275	00
20	14	RES	276	00
21	15	NL	277	10
22	16	BS	278	04
23	17	IL	279	00
24	18		280	00
25	19		281	00
26	1A		282	00
27	1B		283	00
28	1C		284	00
29	1D		285	00
30	1E		286	00
31	1F		287	00
32	20	DS	288	00
33	21	SOS	289	00
34	22	FS	290	00
35	23		291	00
36	24	BYP	292	00
37	25	LF	293	00
38	26	ECB	294	08
39	27	PRE	295	00
40	28		296	00
41	29		297	00
42	2A	SM	298	00
43	2B		299	00
44	2C		300	00
45	2D		301	00
46	2E		302	00
47	2F		303	00
48	30		304	00
49	31		305	00
50	32		306	00
51	33		307	00
52	34	PN	308	00
53	35	RS	309	00
54	36	UC	310	00
55	37	EOT	311	00
56	38		312	00
57	39		313	00
58	3A		314	00
59	3B		315	00
60	3C		316	00
61	3D		317	00
62	3E		318	00
63	3F		319	00
64	40	SP	320	00
65	41		321	00
66	42		322	00
67	43		323	00
68	44		324	00
69	45		325	00

Table 20. Prototype input character translation table (part 3 of 6)

Translation Entry			Function Code	
Byte (Decimal)	Code (Hexadecimal)	Character	Byte (Decimal)	Input Code (Hexadecimal)
70	46		326	00
71	47		327	00
72	48		328	00
73	49		329	00
74	4A		330	00
75	4B	.	331	00
76	4C	<	332	00
77	4D	(	333	00
78	4E	+	334	00
79	4F	|	335	00
80	50	&	336	00
81	51		337	00
82	52		338	00
83	53		339	00
84	54		340	00
85	55		341	00
86	56		342	00
87	57		343	00
88	58		344	00
89	59		345	00
90	5A	!	346	00
91	5B	$	347	00
92	5C	*	348	00
93	5D	)	349	00
94	5E	;	350	00
95	5F	¬	351	00
96	60	-	352	00
97	61	/	353	00
98	62		354	00
99	63		355	00
100	64		356	00
101	65		357	00
102	66		358	00
103	67		359	00
104	68		360	00
105	69		361	00
106	6A		362	00
107	6B	,	363	00
108	6C	%	364	00
109	6D	_	365	00
110	6E	>	366	00
111	6F	?	367	00
112	70		368	00
113	71		369	00
114	72		370	00
115	73		371	00
116	74		372	00
117	75		373	00
118	76		374	00
119	77		375	00
120	78		376	00
121	79		377	00
122	7A	:	378	00
123	7B	#	379	0C
124	7C	@	380	00
125	7D	'	381	00
126	7E	=	382	00

Table 20. Prototype input character translation table (part 4 of 6)

Translation Entry			Function Code	
Byte (Decimal)	Code (Hexadecimal)	Character	Byte (Decimal)	Input Code (Hexadecimal)
127	7F	"	383	00
128	80		384	00
129	81	a	385	00
130	82	b	386	00
131	83	c	387	00
132	84	d	388	00
133	85	e	389	00
134	86	f	390	00
135	87	g	391	00
136	88	h	392	00
137	89	i	393	00
138	8A		394	00
139	8B		395	00
140	8C		396	00
141	8D		397	00
142	8E		398	00
143	8F		399	00
144	90		400	00
145	91	j	401	00
146	92	k	402	00
147	93	l	403	00
148	94	m	404	00
149	95	n	405	00
150	96	o	406	00
151	97	p	407	00
152	98	q	408	00
153	99	r	409	00
154	9A		410	00
155	9B		411	00
156	9C		412	00
157	9D		413	00
158	9E		414	00
159	9F		415	00
160	A0		416	00
161	A1		417	00
162	A2	s	418	00
163	A3	t	419	00
164	A4	u	420	00
165	A5	v	421	00
166	A6	w	422	00
167	A7	x	423	00
168	A8	y	424	00
169	A9	z	425	00
170	AA		426	00
171	AB		427	00
172	AC		428	00
173	AD		429	00
174	AE		430	00
175	AF		431	00
176	B0		432	00
177	B1		433	00
178	B2		434	00
179	B3		435	00
180	B4		436	00
181	B5		437	00
182	B6		438	00
183	B7		439	00

Table 20. Prototype input character translation table (part 5 of 6)

Translation Entry			Function Code	
Byte (Decimal)	Code (Hexadecimal)	Character	Byte (Decimal)	Input Code (Hexadecimal)
184	B8		440	00
185	B9		441	00
186	BA		442	00
187	BB		443	00
188	BC		444	00
189	BD		445	00
190	BE		446	00
191	BF		447	00
192	C0		448	00
193	C1	A	449	00
194	C2	B	450	00
195	C3	C	451	00
196	C4	D	452	00
197	C5	E	453	00
198	C6	F	454	00
199	C7	G	455	00
200	C8	H	456	00
201	C9	I	457	00
202	CA		458	00
203	CB		459	00
204	CC		460	00
205	CD		461	00
206	CE		462	00
207	CF		463	00
208	D0		464	00
209	D1	J	465	00
210	D2	K	466	00
211	D3	L	467	00
212	D4	M	468	00
213	D5	N	469	00
214	D6	O	470	00
215	D7	P	471	00
216	D8	Q	472	00
217	D9	R	473	00
218	DA		474	00
219	DB		475	00
220	DC		476	00
221	DD		477	00
222	DE		478	00
223	DF		479	00
224	E0		480	00
225	E1		481	00
226	E2	S	482	00
227	E3	T	483	00
228	E4	U	484	00
229	E5	V	485	00
230	E6	W	486	00
231	E7	X	487	00
232	E8	Y	488	00
233	E9	Z	489	00
234	EA		490	00
235	EB		491	00
236	EC		492	00
237	ED		493	00
238	EE		494	00
239	EF		495	00
240	F0	0	496	00

Table 20. Prototype input character translation table (part 6 of 6)

Translation Entry			Function Code	
Byte (Decimal)	Code (Hexa-decimal)	Character	Byte (Decimal)	Input Code (Hexadecimal)
241	F1	1	497	00
242	F2	2	498	00
243	F3	3	499	00
244	F4	4	500	00
245	F5	5	501	00
246	F6	6	502	00
247	F7	7	503	00
248	F8	8	504	00
249	F9	9	505	00
250	FA		506	00
251	FB		507	00
252	FC		508	00
253	FD		509	00
254	FE		510	00
255	FF		511	00

Table 21. Prototype output character translation table (part 1 of 5)

Translation Entry			Function Code	
Byte (Decimal)	Code (Hexa-decimal)	Character	Byte (Decimal)	Input Code (Hexadecimal)
0	0		256	20
1	1		257	20
2	2		258	20
3	3		259	20
4	4	PF	260	20
5	5	HT	261	14
6	6	LC	262	00
7	7	DEL	263	20
8	8		264	20
9	9		265	20
10	A		266	20
11	B		267	20
12	C		268	20
13	D		269	20
14	E		270	20
15	F		271	20
16	10		272	20
17	11		273	20
18	12		274	20
19	13		275	20
20	14	RES	276	04
21	15	NL	277	18
22	16	BS	278	1C
23	17	IL	279	00
24	18		280	20
25	19		281	20
26	1A		282	20
27	1B		283	20
28	1C		284	20
29	1D		285	20
30	1E		286	20

Table 21. Prototype output character translation table (part 2 of 5)

Translation Entry			Function Code	
Byte (Decimal)	Code (Hexadecimal)	Character	Byte (Decimal)	Input Code (Hexadecimal)
31	1F		287	20
32	20	DS	288	20
33	21	SOS	289	20
34	22	FS	290	20
35	23		291	20
36	24	BYP	292	08
37	25	LF	293	30
38	26	ECB	294	20
39	27	PRE	295	0C
40	28		296	20
41	29		297	20
42	2A	SM	298	20
43	2B		299	20
44	2C		300	20
45	2D		301	20
46	2E		302	20
47	2F		303	20
48	30		304	20
49	31		305	20
50	32		306	20
51	33		307	20
52	34	PN	308	20
53	35	RS	309	20
54	36	UC	310	00
55	37	EOT	311	20
56	38		312	20
57	39		313	20
58	3A		314	20
59	3B		315	20
60	3C		316	20
61	3D		317	20
62	3E		318	20
63	3F		319	20
64	40	SP	320	28
65	41		321	20
66	42		322	20
67	43		323	20
68	44		324	20
69	45		325	20
70	46		326	20
71	47		327	20
72	48		328	20
73	49		329	20
74	4A		330	00
75	4B	.	331	00
76	4C	<	332	00
77	4D	(	333	00
78	4E	+	334	00
79	4F	|	335	00
80	50	&	336	00
81	51		337	20
82	52		338	20
83	53		339	20
84	54		340	20
85	55		341	20
86	56		342	20
87	57		343	20

Table 21. Prototype output character translation table (part 3 of 5)

Translation Entry			Function Code	
Byte (Decimal)	Code (Hexadecimal)	Character	Byte (Decimal)	Input Code (Hexadecimal)
88	58		344	20
89	59		345	20
90	5A	!	346	00
91	5B	$	347	00
92	5C	*	348	00
93	5D	)	349	00
94	5E	;	350	00
95	5F	¬	351	00
96	60	-	352	00
97	61	/	353	00
98	62		354	20
99	63		355	20
100	64		356	20
101	65		357	20
102	66		358	20
103	67		359	20
104	68		360	20
105	69		361	20
106	6A		362	20
107	6B	,	363	00
108	6C	%	364	00
109	6D	_	365	00
110	6E	>	366	00
111	6F	?	367	00
112	70		368	00
113	71		369	20
114	72		370	20
115	73		371	20
116	74		372	20
117	75		373	20
118	76		374	20
119	77		375	20
120	78		376	20
121	79		377	20
122	7A	:	378	00
123	7B	#	379	00
124	7C	@	380	00
125	7D	'	381	00
126	7E	=	382	00
127	7F	"	383	00
128	80		384	20
129	81	a	385	00
130	82	b	386	00
131	83	c	387	00
132	84	d	388	00
133	85	e	389	00
134	86	f	390	00
135	87	g	391	00
136	88	h	392	00
137	89	i	393	00
138	8A		394	20
139	8B		395	20
140	8C		396	20
141	8D		397	20
142	8E		398	20
143	8F		399	20
144	90		400	20

Table 21. Prototype output character translation table (part 4 of 5)

Translation Entry			Function Code	
Byte (Decimal)	Code (Hexa-decimal)	Character	Byte (Decimal)	Input Code (Hexadecimal)
145	91	j	401	00
146	92	k	402	00
147	93	l	403	00
148	94	m	404	00
149	95	n	405	00
150	96	o	406	00
151	97	p	407	00
152	98	q	408	00
153	99	r	409	00
154	9A		410	20
155	9B		411	20
156	9C		412	20
157	9D		413	20
158	9E		414	20
159	9F		415	20
160	A0		416	20
161	A1		417	20
162	A2	s	418	00
163	A3	t	419	00
164	A4	u	420	00
165	A5	v	421	00
166	A6	w	422	00
167	A7	x	423	00
168	A8	y	424	00
169	A9	z	425	00
170	AA		426	20
171	AB		427	20
172	AC		428	20
173	AD		429	20
174	AE		430	20
175	AF		431	20
176	B0		432	20
177	B1		433	20
178	B2		434	20
179	B3		435	20
180	B4		436	20
181	B5		437	20
182	B6		438	20
183	B7		439	20
184	B8		440	20
185	B9		441	20
186	BA		442	20
187	BB		443	20
188	BC		444	20
189	BD		445	20
190	BE		446	20
191	BF		447	20
192	C0		448	20
193	C1	A	449	00
194	C2	B	450	00
195	C3	C	451	00
196	C4	D	452	00
197	C5	E	453	00
198	C6	F	454	00
199	C7	G	455	00
200	C8	H	456	00
201	C9	I	457	00

Table 21. Prototype output character translation table (part 5 of 5)

Translation Entry			Function Code	
Byte (Decimal)	Code (Hexa-decimal)	Character	Byte (Decimal)	Input Code (Hexadecimal)
202	CA		458	20
203	CB		459	20
204	CC		460	20
205	CD		461	20
206	CE		462	20
207	CF		463	20
208	D0		464	20
209	D1	J	465	00
210	D2	K	466	00
211	D3	L	467	00
212	D4	M	468	00
213	D5	N	469	00
214	D6	O	470	00
215	D7	P	471	00
216	D8	Q	472	00
217	D9	R	473	00
218	DA		474	20
219	DB		475	20
220	DC		476	20
221	DD		477	20
222	DE		478	20
223	DF		479	20
224	E0		480	20
225	E1		481	20
226	E2	S	482	00
227	E3	T	483	00
228	E4	U	484	00
229	E5	V	485	00
230	E6	W	486	00
231	E7	X	487	00
232	E8	Y	488	00
233	E9	Z	489	00
234	EA		490	20
235	EB		491	20
236	EC		492	20
237	ED		493	20
238	EE		494	20
239	EF		495	20
240	F0	0	496	00
241	F1	1	497	00
242	F2	2	498	00
243	F3	3	499	00
244	F4	4	500	00
245	F5	5	501	00
246	F6	6	502	00
247	F7	7	503	00
248	F8	8	504	00
249	F9	9	505	00
250	FA		506	20
251	FB		507	20
252	FC		508	20
253	FD		509	20
254	FE		510	20
255	FF		511	20

Some characters that appear in Tables 20 and 21 have special functions. The definitions of these functions are presented below.

1. Control characters (in order of appearance)

PF	Punch Off	DS	Digit Select	PRE	Prefix
HT	Horizontal Tab	SOS	Start of Significance	SM	Set Mode
LC	Lowercase			PN	Punch On
DEL	Delete	FS	Field Separator	RS	Reader Stop
RES	Restore	BYP	Bypass	UC	Uppercase
NL	New Line	LF	Line Feed	EOT	End of Transmission
BS	Backspace	ECB	End of Block	SP	Space
IL	Idle				

2. Special graphic characters (in order of appearance)

Cent Sign	* Asterisk	> Greater-than Sign
. Period, Decimal Point	) Right Parenthesis	? Question Mark
< Less-than Sign	; Semicolon	: Colon
(Left Parenthesis	¬ Logical NOT	# Pound Sign
+ Plus Sign	- Minus Sign, Hyphen	@ "At" Sign
| Logical OR	/ Slash	' Prime, Apostrophe
& Ampersand, Logical AND	, Comma	= Equal Sign
! Exclamation Point	% Percent Sign	" Quotation Mark
$ Dollar Sign	_ Underscore	

CHARACTER SWITCH TABLE

The table of miscellaneous control characters and the translation tables described above make up the character switch table. The table of miscellaneous control characters includes:

Source list EOB character
: defines the end of an input block. Its initial value is X'26'. (Do not change the value.) This character should not be used as input within a command statement.

Command system continuation character
: indicates that a line is being continued. 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 is continued past the carriage return. Initially, this character is a hyphen (X'60').

Command system break character
: tells the system that a command follows. Initially, this character is an underscore (X'6D').

Transient statement prefix character
: is an indicator that whatever follows is sent to a predetermined entry point for execution. Initially, this character is a vertical stroke (X'4F').

Concatenation character
: indicates that the next line should be concatentated with this line. This character must be the last character of a line of data for the text editor, and the CONREC implicit operand must be set to Y. The system-supplied value for the concatenation character is colon (X'7A'). For more information, see "Concatenating Input Records" in Section 2 of Part II.

System scope mask
: controls searches for explanatory messages issued by the user prompter. Its default value is X'29'. The use of the system scope

mask is explained under "Message File Construction" in Section 5 of Part II.

User scope mask
works in the same way as the system scope mask works, but on user-created messages in the USERLIB. The user may set this mask according to his own search logic X'29'. The default value is X'29'.

Command prompt string
is issued by the system and requests that a command be entered. This may be a string of up to eight characters. The initial default is an underscore followed by a backspace and a carriage-return suppression character (colon).

SYSIN keyboard/card reader 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 does not recognize his subsequent specification of a card reader as the input device. The initial value is E.

Carriage return suppression character
indicates that carriage return is suppressed when it is the last character in a message being written to SYSOUT by the command system. In this case, the system does not add a new-line character to the text. The system-supplied value is a colon (X'7A').

Tables 22-25 contain control codes and characters that can be used for formatting printed output and for selecting stackers for punched output.

Table 22. Printer 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.

Table 23. FORTRAN control characters* for the printer

Function	Character
Skip no lines before printing	+
Skip 1 line before printing	blank
Skip 2 lines before printing	0
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

* FORTRAN control characters are defined by American National Standard FORTRAN, ANSI X3.9-1966.

Table 24. IBM 2540 punch machine codes

Function	EBCDIC	Column Binary
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

Table 25. FORTRAN control characters for the punch

Function	Character
Select punch pocket 1	V
Select punch pocket 2	W

This appendix describes special use of the DDEF command. To define typical public data sets, see the DDEF command description in Part III. A typical public data set is created with the virtual access method (VAM), is sequential or indexed sequential and resides on direct access public storage.

Table 26 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 shown in Table 26.

Table 26. Format illustration of the DDEF command

```
┌─────────┬──────────────────────────────────────────────────────────────┐
│Operation│Operand                                                       │
├─────────┼──────────────────────────────────────────────────────────────┤
│DDEF     │DDNAME=data definition name                                   │
│         │[,DSORG={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={A|W}]                   │
│         │       [,LRECL=record length]                                 │
│         │       [,BLKSIZE=block length][,KEYLEN=key length]            │
│         │       [,DEN=tape density]                                    │
│         │       [,TRTCH=data conversion]                               │
│         │       [,EROPT=error option][,PAD=padding]                    │
│         │       [,RKP=key displacement]                                │
│         │       [,IMSK=error recovery procedures]                      │
│         │       [,BUFOFF=n])                                           │
│         │                                                              │
│         │         DA[,direct access type]                              │
│         │ ,UNIT=( TA[,tapedevice type]     )                           │
│         │         device                                               │
│         │                                                              │
│         │[,SPACE=({CYL|TRK|record length},primary[,secondary][,HCLD])  │
│         │                                                              │
│         │[,VOLUME=([volseqno|PUBLIC|PRIVATE],{volserno|PRIVATE},       │
│         │   [volserno|PRIVATE],...)]                                   │
│         │                                                              │
│         │[,LABEL=([file sequence number][,{NL|SL|SUL|AL|AUL}]          │
│         │   [,RETPD=retention period])]                                │
│         │                                                              │
│         │[,DISP={MOD|CLD|NEW}]                                         │
│         │                                                              │
│         │[,OPTION={CONC|JOBLIB}]                                       │
│         │                                                              │
│         │[,RET=retention code]                                         │
│         │                                                              │
│         │[,PROTECT={Y|N}]                                              │
└─────────┴──────────────────────────────────────────────────────────────┘
```

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

Specified as: From one to eight alphameric characters, the first of which must be alphabetic. To define the PCSOUT data set for the DUMP command, PCSOUT should be given as the data definition name. DDNAME may not begin with SYS; these characters are used to prefix system-reserved data definition names.

Note: When DDEF is used by an assembler language user, the name specified for DDNAME must be the same as the name specified for the DDNAME operand in the DCB macro instruction. When DDEF is employed by a FORTRAN user, the name specified for DDNAME must be "FTxx-Fyyy", where "xx" is the two-digit data set reference number, and "yyy" is the three-digit file sequence number in his FORTRAN program.

DSORG

indicates the organization of the data set being defined. (See Table 27.)

Specified as:

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.

Table 27. Data set organization requirements (part 1 of 2)

Data Set Characteristics	DSORG* PS	VI	VP	VS	Comments
Any data set on a public volume		x	x	x	
Any data set on a 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 a 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	
Data Sets Created by or Used by Language-Processing Commands					
Source data set for language processing		x			Line data set only; if source data sets are entered from the terminal, a line data set is built automatically.
Source statements stored as part of the SYSIN data set		x		x	Line data set will be built from source statements.
Object module produced by the language processor				x	Object module automatically becomes a member of the most recently defined job library, if any, or of the user's library (USERLIB).

Table 27. Data set organization requirements (part 2 of 2)

Data Set Characteristics	DSORG* PS	VI	VP	VS	Comments
Job library			x		
Listing data set produced by the language processor		x			
Data Sets Used for I/O Operations					
PCSOUT data set		x			
Input to the WT Command		x		x	
Input to the PRINT Command	x	x		x	
Input to the PUNCH Command		x		x	
Data Sets for Special Command Usage					
Data set used by the CDD command		x			Line data set only.
Data set used by the LINE? command		x			Line or language processor listing data set only.
Data set created by the DATA command		x	x	x	User option; if VI, it must be line data set. The member may be VS line or VI line.
Data set created by the MODIFY command		x	x		User option determines whether VI is for a line data set. The member must be VI.
Data set created by the text editor		x	x		User option determines whether VI is for a line data set. The member must be VI.

*Note: If one DSORG option is checked, the data set must be that organization. If more than one option is checked, select either organizationl

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 a 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 OS or OS/VS is introduced into TSS for the first time, the name 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.

For ASCII input, you may specify any nonalphameric characters in DSNAME. However, if nonalphameric characters are specified, DSNAME must be preceded by a blank and must be enclosed by single quotation marks. For example:

```
DSNAME= 'dsname'
```

DCB

specifies data control block information.

Detailed descriptions of the DCB suboperands are given in Assembler Programmer's Guide and FORTRAN Programmer's Guide.

Note: If the data set is on tape 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.

UNIT

specifies the type of device required by the data set. Direct access devices may be specified for either public cr 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 and, therefore, may be changed.

DA[,direct access type]
specifies the type of direct access device.

Specified as:

2311 - 2311 disk
2314 - 2314/2319 disk
3330 - 3330-1 disk
333B - 3330-11 disk.

System default: the type of direct access device specified at system generation.

TA[,tape device type]
specifies magnetic tape device is required for the data set.

Specified as:

7 - seven-track tape, data converter not required
7DC - seven-track tape with data converter feature
9D2 - 9-track tape with 800 bpi capability
9D3 - 9-track tape with 1600 bpi capability
9D4 - 9- track tape with 6250 bpi capability

System default: the type of tape device specified at system generation.

device
specifies the symbolic device address of a nonstandard device.

Specified as: a four-digit hexadecimal symbolic device address.

SPACE

indicates the direct access storage allocation for the data set.

Specified as:

CYL
space requirements are expressed as number of cylinders.

TRK
space requirements are expressed as number of tracks.

(record length)
space requirements are expressed as a decimal number that specifies the average length of the physical records; the number must not exceed 32,767.

System default: if the data set organization is QSAM or BSAM (see DSORG), the space requirements are assumed to be expressed in terms of cylinders. If the data set organization is VISAM, VPAM, or VSAM (see DSORG), the space requirements are assumed to be in pages (of 4096 bytes).

Note: this field must be defaulted if the data set organization is VAM.

(primary)
a one- to three-digit decimal number that indicates the amount of space to be allocated initially. This operand may express space in terms of tracks or cylinders or in terms of number of pages.

System default: the primary space allocation specified at system generation is assumed.

(secondary)
a one- to three-digit decimal number specifying the amount of additional space to be allocated each time the space already allocated has been exhausted and more data is to be written. No more than 256 pages are allocated at one time even if a number greater than 256 is specified.

System default: the secondary space allocation specified at system generation.

HOLD
specifies 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 is released when the data set is closed.

Note: If the SPACE operand is not specified, the direct access storage allocation specified at system generation is assigned.

If DISP=OLD, the SPACE operand is ignored.

VOLUME
specifies the volume on which the data set resides. This field must always be used when creating a new data set residing on a private volume or when referring to an existing uncataloged data set residing on a private volume. This field must also be used when expanding an existing private data set. When expanding an existing private cataloged data set, only the new volumes to be added to the data set (PRIVATE or volume serial number) need be referred to. This field is never required for data sets on public volumes. However, this field may be specified for new data sets on public volumes, if only existing public volume serial numbers are specified. Initial space allocation is limited to the specified volumes.

Specified as:

(volseqno)
a one- to four-digit number specifying the volume sequence number of 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.

PUBLIC
indicates that the data set is to be placed on public storage volumes. PUBLIC may not be specified when a volume serial number is used.

PRIVATE
specifies that volumes are to be allocated from the system pool (that is, the scratch or disk available to the operator). Once assigned, the volume remains the user's, exclusively, until he notifies the operator that it can be returned to the pool. The user must use this option to request initial or additional scratch volumes for data sets on private volumes. PRIVATE must not be specified when a volume serial number is used.

(volserno)
from 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 or to specify initial or additional volume serial numbers for data sets on private volumes; it is optional for new data sets that will reside on public volumes. For ASCII tapes with nonalphameric characters in the volume serial number, the volume serial number, preceded by a blank, must be contained in apostrophes.

System default: if VOLSEQNO is specified, the data set is assumed to be cataloged, and the volume serial numbers are retrieved from the catalog. If PRIVATE or PUBLIC is specified, VOLSERNO must be omitted, and a volume serial number is assigned by the system.

Note: VOLUME 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.

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.

Five suboperands specify either the type of labeling desired or the absence of labeling:

NL - no labels

SL - standard labels (as specified at system generation)

SUL - standard and user labels

AL - standard ASCII labels

AUL - standard ASCII and user labels

System default: SL.

The following suboperand can be used to indicate the number of days a data set is to be saved.

RETPD
retention period

Specified as: a four-digit decimal number; this suboperand is applicable for data sets on direct access volumes or labeled tapes.

System default: 0 days.

Note: If the entire LABEL operand is defaulted, the labeling conventions specified at system generation are assumed, unless the data set being defined is already cataloged. If the data set is cataloged already, label information is retrieved from the catalog.

DISP
specifies whether the data set already exists or is to be created.

Specified as:
MOD - the data set being defined exists; an addition to it is being made

OLD - the data set being defined exists

NEW - the data set being defined has not yet been created

Note: MOD causes logical positioning after the last record of the data set.

System default: OLD if the data set is cataloged; NEW if it is not cataloged.

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 concatenation of data sets indicated by the DDNAME operand.

Specified as:

CONC
only SAM data sets that are not job libraries can be concatenated with one or more data sets having the same DDNAMEs. The order of access for 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 name specified in the DSNAME operand is entered into the program library list.

RET
specifies the catalog attributes to be assigned to a VAM data set.

Specified as: P or T, C or L, or U or R.

P -- permanent storage
T -- temporary storage
C -- erase at CLOSE
L -- erase at LOGOFF
U -- unlimited access
R -- read-only access

System default: PU is assumed; when T is specified, LU is assumed; when P is specified, U is assumed.

Note: A data set is not erased at logoff if a RELEASE command has been issued for it and the retention code has been specified as RET=T.

PROTECT
specifies whether to mount the tape with or without the file-protect ring.

Specified as:

Y - mount tape with ring out.
N - mount tape with ring in.

System default: If DISP is NEW or MOD, the tape is mounted with a ring inserted. If DISP is OLD, there is no default; the decision depends on the installation's operational procedure.

Functional Description: The DDEF command causes a system entry tc be established for the data set definition. The link between this definition and the problem program's reference to the data set (through the DCB) is the DDNAME. The entry containing the data set 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 for device allocation and volume mounting when the defined data set is private and resides on a demountable volume such as a reel of tape or a disk pack. A request for a private device will not be fulfilled if the user has exceeded his ration.

Programming Notes: The DDEF command that defines a cataloged data set is brief and simple. The required operands are DDNAME and DSNAME. DSORG is nct necessary because the organization of the data set is described in its catalog entry. Other operands are unnecessary.

DDEF commands that define uncataloged data sets may be divided into two groups: those defining data sets that are generated during execution of the program, but do not yet exist; and those defining 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, DSORG, SPACE, and LABEL operands. Exactly which fields he uses, other than the required DDNAME and DSNAME, depends on the characteristics 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. 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.

To change the DDNAME assigned in a previous DDEF command, the user must issue a DDEF command with a new DDNAME and the same DSNAME that was previously specified. Any other operands entered are ignored. If the user wants to change the other parameters, he must issue a RELEASE command to delete the previously issued DDEF command and re-issue the DDEF command to establish a new system entry for the data set definition. Table 28 summarizes some operations for which the DDEF command is used.

Table 28. Typical use of DDEF operands

Operation	DDNAME	DSORG	DSNAME	DISP	DCB	UNIT	SSI ACE	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]			
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 concatenated data set except the first in concatenation)	X	X	X	X		[X]		[X]	[X]	

Note: [] indicates that the operand may be used, but is not mandatory.

The following examples show one way of entering a DDEF command to get the specified operation performed. In each example, DDNAME, DSORG, and DSNAME are specified positionally; the other operands are given in keyword notation.

1. Read a cataloged data set:

```
ddef ddn,test1
```

2. Read an uncataloged data set:

```
ddef ddn1,ps,test2,unit=(da,2311),volume=(,012300),disp=new
```

3. Write a data set on a public volume:

```
ddef ddn2,vp,test3
```

4. Write a data set on a private volume:

```
ddef ddn3,ps,test4,unit=(ta,9),volume=(private)
```

   The VOLUME operand could be entered as:

```
volume=(,005431)
```

5. Modify any data set of a private volume:

```
ddef ddn4,ps,test5,unit=(ta,9),volume=(,012301),disp=mod
```

6. Concatenate cataloged data sets while reading private volumes:

```
ddef ddn6,ps,test6,disp=old
ddef ddn6,ps,test7,disp=old,option=conc
ddef ddn6,ps,test8,disp=old,option=conc
```

The DDEF command also has several special uses. Among them are:

1. Define a job library:

```
ddef dd1,vp,dsn1,option=joblib
```

   No other operands are necessary. If the data set already exists, it is defined as a job library; if it does not exist, a new job library is created.

2. Define a data set for dumps. Mandatory operands must be given.

```
ddef pcsout,VI,dump1
```

To complete the DCB of a data set at execution, include the DCB operand. Other operands are included as needed for the particular data set.

To concatenate data sets, use the CPTION=CONC operand. 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 to be concatenated, except for the first-defined data set in the concatenation. Each of the remaining data sets in the concatenation must have the same DDNAME as the first-defined data set.

APPENDIX F: CURRENT LINE POINTER

After a text-editing command has been executed, the current line pointer (CLP) is positioned according to certain general rules:

- If the command is canceled by the system, CLP is unchanged.
- If the N2 value is equal to the last record in the data set or region, CLP is set to the value of N2 plus the value of INCR.
- If the N2 value is not equal to the last record, CLP is set to the line after N2.

For each text-editing command, CLP is positioned as follows:

CONTEXT
to the line following the last line searched (N2); if N2 is the last line, CLP is set to N2 plus the value of INCR.

CORRECT
to the line after N2; if N2 is the last line, CLP is set to N2 plus the value of INCR.

DISABLE, ENABLE, POST, STET
CLP is not changed.

EDIT
to the first line in an existing data set; to the value of BASE (the system default is 100) for a new data set.

EXCISE
to the value specified by N1.

EXCERPT
to the value of the last line inserted plus the value of INCR or to the next-existing line number, whichever is less.

INSERT
to the line number of the last data line entered plus the value of INCR. If this exceeds the next-existing line number, CLP is set to that line number. If no data lines are entered, CLP is set to N1.

LIST
to the line after N2; if N2 = last line, CLP = N2 + value of INCR.

LOCATE
to the record containing the search string if the string is found; if the string is not found, CLP is set to the line following N2. If N2 is the last line, CLP is set to N2 plus the value of INCR.

NUMBER
to the last line renumbered plus the value of INCR or to the next-existing line, whichever is less.

REGION
to the first line in region if the region exists. If the region is not currently in the data set, CLP is set to the new region name and a line number specified by BASE (system default is 100).

REVISE
to the value of the last data line entered plus the value of INCR. If this exceeds the next-existing line number, CLP is set to that line number. If no data lines are entered, CLP is set to N1.

UPDATE
CLP is not changed.

APPENDIX G: COMMAND FORMATS

Table 29 summarizes the command formats that appear in Part III. This appendix can be used as a reference to the format of a command if you do not need the detail presented in Part III.

Table 29. Command format summary (part 1 of 6)

Operation	Operand
ABEND	
ABENDREG	
ASM	NAME=module name[,STORED={Y|N}] [,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|E}] [,STEDIT={Y|N}][,ISDLIST={Y|N}][,PMDLIST={Y|N}] [,LISTDS={Y|N}][,LINCR=(first line number,increment)]
AT	instruction location[,...]
BACK	DSNAME=data set name
BEGIN	application name[,application parameters]
BLIP	TIME=,*READ=
BLIP?	
BRANCH	INSTLOC=instruction location
BUILTIN	NAME=command name[,EXTNAME=bpkd macro name] [,PROLIB=data set name]
C	
CA	
CALL	[NAME=entry point name][,module parameters]
CANCEL	BSN=batch sequence number
CATALOG (Form 1)	DSNAME=current data set name[,STATE={N|U}][,ACC={R|U}] [,NEWNAME=new data set name]
CATALOG (Form 2)	GDG=generation data group name,GNO=number of generations [,ACTION={A|O}][,ERASE={Y|N}]
CB	
CDD	DSNAME=data set name, {data definition name|(data definition name[,...]
CDS	DSNAME1=input data set name[(member name[,...])], DSNAME2=copy data set name[(member name)] [,ERASE={Y|N}][,COPYBASE=first line number, COPYINCR=increment][,REPLACE={R|I}]
CHGPASS	[NEWPASWD=password]

Table 29. Command format summary (part 2 of 6)

Operation	Operand
CLOSE	[DSNAME=data set name][,TYPE=T] [,DDNAME=data definition name]
COBOL	NAME=modulename [,OSOPTS=(opt1,opt2,...)] [,SOURCEDS=sourcedsname]
CONTEXT	[N1=starting position][,N2=ending position] ,STRING1=search string[,STRING2=replacement string]
CORRECT	[N1=starting line][,N2=ending line][,SCOL=starting column] [,CORMARK=replacement correction characters][,CHAR={C|M|H}]
DATA	DSNAME=data set name ,RTYPE={I|LINE|FTN|CARD|S} [,DBASE=first line number] [,DINCR=increment]
DDEF	DDNAME=data definition name[,DSORG={VI|VS|VP}] ,DSNAME=data set name
DDNAME?	[JOBLIB={Y|N}]
DEFAULT	{operand=[value]}[,...]
DELETE	[DSNAME=data set name]
DISABLE	
DISPLAY	data field name or expression[,...] id? data field name or expression[,...]
DMPRST	FROMDEV={2311|2314|24xx|3330|333B},FRVOLID={volid|(volid [,volid])},TODEV={2311|2314|24xx|3330|333B} [,TOVOLID={volid|(volid[,volid])|PRIVATE}] [,NEWVLID=volid][,WRITCHK={YES|NO}][,LABEL={RETAIN|NO}] [,],RUNMODE={BACK|FORE}
DSS?	NAMES= data set name|(data set name[,...]
DUMP	data field name or expression[,...] id? data field name or expression[,...]
EDIT	DSNAME=data set name[(member name)][,RNAME=region name] [,REGSIZE=region name length]
EJECT	
ENABLE	
END	
ERASE	[DSNAME=data set name[(member name)]][,SHARED={Y|N}]
EVV	DEVICE={2311|2314|3330|333B} ,VOLUME=(volume serial number [,...])
EXCERPT	DSNAME=data set name[(member name)][,RNAME=region name] [,N1=starting line[,N2=ending line]]
EXCISE	[N1=starting line][,N2=ending line]

Table 29. Command format summary (part 3 of 6)

Operation	Operand
EXECUTE	DSNAME=data set name
EXHIBIT	OPTION1={BWQ[,TYPE={ALL|BSN.number}]| UID[,TYPE={CONV|BACK|UID.userid|ALL}]}
EXIT	[SIRTEST={Y|N}]
EXPLAIN	MSGID|ORIGIN|word|TEXT|RESPONSE| [,message identification]|MSGE|MSGS}
FILEDEF	DDNAME=ddname,DSORG=VI|VS|VP[,DSNAME=dsname...] [,MACRO=CONC] [,OSDDN=osddname] [,OSKEYLE=number]
FILEREL	OSDDN=osddname
FTN	NAME=module name[,STORED={Y|N}][,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}] [,LISTDS={Y|N}][,LINCR=(first line number,increment)]
FTNH	NAME=modulename [,OSOPTS=(opt1,opt2,...)] [,SOURCEDS=sourcedsname]
GAV	[TYPE={SYN|DEF|CSW}]
GDV	DFLT=term
GO	
GOTO	{command|OUT|'comment'}
GSV	NAME=value or term [,SEARCH={T|V}]
HASM	NAME=module name[,OSOPTS=(opt1,opt2,...)] [,SOURCEDS=sourcedsname]
IF	condition
INSERT	[N1=starting line][,INCR=increment]
JOBLIBS	DDNAME=data definition name
K	
KA	
KB	
KEYWORD	[PROCNAME=command name]
LINE?	DSNAME=data set name[(member name)] [,{line number|(first line number,last line number)}[,...]]
LIST	[N1={starting position|CLP}][,N2=ending position] [,CHAR={C|H|M}]
LL	LGH=,*TRUNCATE=,*RESET=

Table 29. Command format summary (part 4 of 6)

Operation	Operand
LNK	NAME=module name[,STORED={Y|N}] [,LIB=data definition name of library] [,VERID=version identification][,ISD={Y|N}][,PMDLIST={Y|N}] [,LISTDS={Y|N}][,LINCR=(first line number,increment)]
LOAD	[NAME=entry point name]
LOCATE	[N1=starting position][,N2=ending position] [,STRING=character string]
LOGOFF	
LOGON	user identification[,password][,addressing][,charge number] [,control section packing][,maximum auxiliary storage] [,pristine][,user IVM code]
LTDS	
MCAST	[EOB=end-of-block character] [,CONT=continuation character] [,CLP=break character] [,TRP=transient statement prefix character] [,RCC=concatenation character] [,SSM=system scope mask] [,USM=user scope mask] [,KC=keyboard/card reader character] [,RS=carriage return suppression character] [,CP=command-prompt string]
MCASTAB	[INTRAN={Y|N}][,OUTRAN={Y|N}]
MODIFY	SETNAME=data set name[,CONF=R][,LRECL=record length, KEYLEN=key length,RKP=key displacement,RECFM={V|F}] [,FTN={Y|N}]
NUMBER	[N1=starting line][,N2=ending line][,NBASE=base number] [,INCR=increment]
ODC	ODCMOD=module[,ODCPLI=Y|N] [,ODCERASE=Y|N]
OSDD?	
OSRUN	module[,' parm']
PC?	NAMES={data set name|(data set name[,...])}
PERMIT	DSNAME={data set name|*ALL} [,USERID={(user identification[,...])|*ALL}] [,ACCESS={R|RO|RW|U}]
PLI	[NAME=module name][,PLIOPT=compiler option list] [,PLCOPT=language controller options] [,SOURCEDS=source data set name] [,MERGELST=converter input list] [,MERGEDS=converter input data set] [,MACRODS=intermediate data set name] [,EXPLICIT=external names to be changed] [,XFERDS=transfer vector data set name]

Table 29. Command format summary (part 5 of 6)

Operation	Operand
PLIOPT	NAME=modulename[,OSOPTS=(opt1,opt2,...)] [,SOURCEDS=sourcedsname]
POD?	[PODNAME=data set name][,DATA=Y][,ALIAS=Y] [,MODULE={module name|*ALL}]
POST	
PRINT	DSNAME=data set name[,STARTNO=starting position] [,ENDNO=ending position] ,PRTSP=EDIT|1|2|3| [,HEADER=H] [,LINES=lines per page] [,PAGE=P] [,ERASE={Y|N}][,ERROROPT={ACCEPT|SKIP|END}] [,FORM=paper form][,STATION=station id]
PRMPT	MSGID=message identification [,INSERTn=inserted character[,...]]
PROCDEF	NAME=procedure name[,PROLIB=data set name]
PROFILE	[CSW={N|Y}]
PUNCH	DSNAME=data set name[,] [,STARTNO=starting position][,ENDNO=ending position] [,STACK={1|2|3|EDIT}][,ERASE={Y|N}][,FORM=card form]
PUSH	[SIRTEST={Y|N}]
QUALIFY	MNAME=[link-edited module name.]object module name
REGION	[RNAME=region name]
RELEASE	DDNAME=data definition name[,DSNAME=data set name] [,{SCRATCH|HOLD}][,{SCRATCH|HOLD}]
REMOVE	{statement number[,...]|ALL}]
RET	DSNAME=data set name,RET=retention code
REVISE	[N1=starting line][,N2=ending line][,INCR=increment]
RTRN	
SECURE	(TA=number of devices[,type of device]) [,...] (DA=number of devices[,type of device])
SET	{data location=value}[,...]
SHARE	DSNAME=data set name,USERID=owner's user identification [,OWNERDS={owner's data set name|*ALL}]
SPACE	NUMLINES=(number lines to space)
STACK	
STET	
STOP	

Table 29. Command format summary (part 6 of 6)

Operation	Operand
STRING	
SYNONYM	[term=[value]}],...]
TIME	[MINS=minutes]
TRANSLAT	TYPE,FROM,TO,USN,CP
TRAP	FETCH|STORE|REF},[location[:location}] GR,{nR,...|nR:nR} BRANCH[,LOCATION[:LOCATION}[,LOCATION[:LOCATION}]}
TV	DSNAME1=tape data set name[,DSNAME2=VAM data set name] [,OVERLAY=Y|N][,RETAIN=Y|N][,FROMID=USER IDENT] [,TOID=USER IDENT]
UNLOAD	[NAME=entry point name]
UPDATE	
USAGE	
VT	DSNAME1=VAM data set name[,DSNAME2=tape data set name] [,ERASEDS1=Y|N][,RETAIN=Y|N] [,FROMID=USER IDENT][,TOID=USER IDENT] [,CATDS2=Y|N]
VV	DSNAME1=current data set name[,DSNAME2=new data set name] [,ERASEDS1=Y|N][,OVERLAY=Y|N][,RETAIN=Y|N] [,FROMID=USER IDENT][,TOID=USER IDENT]
WT	DSNAME=current data set name,DSNAME2=tape data set name [,VOLUME=tape volume number][,FACTOR=blocking factor] [,STARTNO=starting position][,ENDNO=ending position] ,PRTSP={EDIT|1|2|3} [,HEADER=H][,LINES=lines per page][,PAGE=P] [,ERASE={Y|N}]
ZLOGON	

APPENDIX H: KEY TO VALUES DISPLAYED BY USAGE COMMAND

Table 30 summarizes the output of the USAGE command. The field abbreviations are defined and the meanings of the statistics associated with each field are given.

Table 30. Explanation of output from the usage command (part 1 of 2)

Field Abbreviation	Statistic		
	Ration	Current Usage	Accumulative Usage
TEMP STOR (temporary storage) PERM STOR (permanent storage)	Number of pages available for this user's data sets.	Number of pages currently occupied by this user's data sets.	Number of page-seconds utilized in storing this user's data sets since the accumulative statistic was last set to 0; calculated by summing the time (in seconds) each page was, or has been, assigned to the user.
DA DEV (direct access devices) MAG TAP (magnetic tape) PRINTERS (high-speed printers) RD-PU (card readers and card punches)	Number of devices of this type available to this user.	Number of devices of this type currently assigned to this user.	Number of device-seconds utilized by this user since the accumulative statistic was last set to 0; calculated by summing the time (in seconds) each device was, or has been, assigned to this user.
TSS TASKS	Maximum number of tasks that can be associated with this USERID.	Number of active tasks currently associated with this USERID.	Not applicable
BULKIN BULKOUT	Not applicable	Not applicable	Total number of bulk input (BULKIN) and bulk output (BULKOUT) tasks associated with this USERID since this accumulative statistic was last set to 0.

Table 30. Explanation of output from the usage command (part 2 of 2)

Field Abbreviation	Statistic		
	Ration	Current Usage	Accumulative Usage
CPU TIME (execution time)	Maximum amount of CPU execution time permitted to tasks associated with this USERID; presented in the form hhh.mm.ss, where hhh is in hours, mm is in minutes, and ss is in seconds.	Time spent executing in the CPU since the current user task was logged on; presented in the form mm.ss.nn, where mm is in minutes, ss is in seconds, and nn is in milliseconds.	Time spent executing in the CPU since this accumulative statistic was last set to 0; presented in the form hhh.mm.ss, where hhh is in hours, mm is in minutes, and ss is in seconds. It is this value that is compared to the maximum amount of CPU execution time permitted (the ration) to see if the user has exceeded his limit.
CONN TIME (connect time)	Maximum amount of time that this user can be connected to the system from a terminal; presented in the form hhh.mm.ss, where hhh is in hours, mm is in minutes, and ss is in seconds.	Time elapsed since the current user task was logged on; presented in the form hhh.mm.ss, where hhh is in hours, mm is in minutes, and ss is in seconds.	Sum of the current times of each terminal session since this accumulative statistic was last set to 0; presented in the form hhh.mm.ss, where hhh is in hours, mm is in minutes, and ss is in seconds; compared to the maximum amount of the connected time in the ration to see if the user has exceeded his limit.

APPENDIX I: PL/I COMPILER OPTIONS

The PLIOPT operand of the PLI command specifies a list of PL/I options to be used by the compiler. The list of compiler options following the equal sign in the PLIOPT operand must be enclosed in parentheses unless only one value is given. Each option must be separated by commas. For an option that includes a numeric specification (for example, SIZE or LINECNT), only significant digits need be specified. Furthermore, for an option that includes more than one numeric specification (for example, SORMGIN), the numbers must be enclosed in parentheses and separated by commas.

There is no required order for specifying the compiler options, but if conflicting options are specified, the last specification in the list is used. A brief explanation of the compiler options follows. This information is summarized in Table 31. The standard defaults are shown in the table, but you can specify an alternative. Additional information appears in PL/I Programmer's Guide.

Table 31. Formats of compiler options, abbreviations, and standard defaults

Category	Compiler Option Format	Abbreviated Name	Standard Default
Control options	OPT=n	O	01
	STMT|NOSTMT	ST|NST	NOSTMT
	OBJNM=aaaaaaaa	N	None
	SYNCHKT|SYNCHKS|SYNCHKE	SKT|SKS|SKE	SYNCHKS SYNCHKE
Preprocessor options	MACRO|NOMACRO	M|NM	NOMACRO
	COMP|NOCOMP	C|NC	COMP
	MACDCK|NOMACDCK	MD|NMD	NOMACDCK
Input options	CHAR60|CHAR48	C60|C48	CHAR60
	BCD|EBCDIC	B|EB	EBCDIC
	SORMGIN=(mmm,nnn,[ccc])	SM	(1,100)
Output options	LOAD|NOLOAD	LD|NLD	LOAD
	DECK|NODECK	D|ND	NODECK
Listing options	LINECNT=xxx	LC	50
	OPLIST|NOPLIST	OL|NOL	OPLIST
	SOURCE2|NOSOURCE2	S2|NS2	SOURCE2
	SOURCE|NOSOURCE	S|NS	SOURCE
	NEST|NONEST	NT|NNT	NONEST
	ATR|NOATR	A|NA	NOATR
	XREF|NOXREF	X|NX	NOXREF
	EXTREF|NOEXTREF	E|NE	NOEXTREF
	LIST|NOLIST	L|NL	NOLIST
	FLAGW|FLAGE|FLAGS	FW|FE|FS	FLAGW
Dummy options	SIZE=yyyyyy|yyyK|999999|MAX	SIZE	MAX
	OBJIN|OBJOUT*	OBJIN|OBJOUT	OBJOUT
	EXTDIC/NOEXTDIC	ED|NED	ED

*Note: Formerly referred to as M91/NOM91

CONTROL OPTIONS

Control options establish the conditions for compilation.

OPT: This option specifies the type of optimization required:

OPT=0
instructs the compiler to keep object-program storage requirements to a minimum at the expense of object-program execution time.

OPT=1
causes object-program execution time to be reduced at the expense of storage.

OPT=2
has the same effect as OPT=1, but in addition requests the compiler to optimize the machine instructions generated for certain types of DO loops and expressions in subscript lists. IBM TSS: PL/I Language Reference Manual includes a discussion of DO loop and subscript-expression optimization.

There is little difference in compilation time for optimization levels 0 and 1, but specifying OPT=2 can increase compilation time.

STMT or NOSTMT: This option requests the compiler to produce additional instructions that allow statement numbers from the source program to be included in diagnostic messages produced during execution of the compiled program. The use of this option increases execution time. However, you can get information about statement numbers and their associated offsets by referring to the table of offsets in the listing.

OBJNM: This option has meaning only in a *PROCESS statement. When the PL1 command is executed this option is ignored and the value is taken from the NAME parameter. The OBJNM option allows you to specify a name (from one to eight alphameric characters) for successive compilations in a batched compilation.

SYNCHKT or SYNCHKS or SYNCHKE: These options allow the user to control the operation of the PL/I compiler when errors are encountered in the "Dictionary" phase of compilation. The effect of each option is as follows:

- SYNCHKT overrides the system default. There is no prompting.
- SYNCHKS causes prompting in conversational mode or termination in nonconversational mode when errors of severity SEVERE are found.
- SYNCHKE causes prompting in conversational mode or termination in nonconversational mode when errors of severity SEVERE or ERROR are found.

The system defaults are SYNCHKS in conversational mode; SYNCHKE in nonconversational mode.

PREPROCESSOR OPTIONS

Preprocessor options request the services of the preprocessor and specify how its output is to be handled.

MACRO or NOMACRO: Specify MACRO when you want to employ the compiler preprocessor.

COMP or NONCOMP: Specify COMP if you want the PL/I source module produced by the preprocessor compiled immediately. The source module is

then read by the compiler from the data set identified by the DDNAME PLIMAC.

MACDCK or NOMACDCK: Specify MACDCK if you want to save the intermediate macro file that has the DDNAME of PLIMAC. NOMACDCK causes the file to be erased after compilation is complete.

INPUT OPTIONS

Input options specify the format of the input to the compiler.

CHAR60 or CHAR48: If the PL/I source statements are written in the PL/I 60-character set, specify CHAR60; if they are written in the 48-character set, specify CHAR48. IBM Time Sharing System: PL/I Language Reference Manual lists both character sets. (The compiler accepts source programs written in either character set if you specify CHAR48. However, use of CHAR48 is inefficient.)

BCD or EBCDIC: The compiler accepts source statements in which the characters are represented by either binary coded decimal (BCD) or extended binary coded decimal interchange code (EBCDIC). Whenever possible, use EBCDIC since BCD requires translation and is less efficient. (See PL/I Language Reference Manual for the EBCDIC representation of both the 48-character set and the 60-character set.)

SORMGIN: This option specifies the extent of the part of each input record that contains the PL/I source statements. (SORMGIN represents source margin.) The compiler does not process data that is outside these limits. The option can also specify the position in the record of a FORTRAN control character. This character is used to format the listing of source statements produced by the compiler if you include the SOURCE option. The format of SORMGIN is:

```
SORMGIN=(mmm,nnn[,ccc])
```

where,

mmm represents the number of the first byte of the field that contains the source statements

nnn represents the number of the last byte of the source statement field

ccc represents the number of the byte that will contain the control character

The value mmm must be less than or equal to nnn, and neither must exceed 100. The value ccc must be outside the limits set by mmm and nnn. The valid FORTRAN control characters are:

blank	Skip one line before printing
0	Skip two lines before printing
-	Skip three lines before printing
+	Suppress space before printing
1	Start new page

The carriage control character can be ignored by specifying zero. Zero is the system default.

OUTPUT OPTIONS

Output options specify the type of data set that will contain the object module.

LOAD or NOLOAD: Specifying LOAD invokes the same action as the DECK option, but in addition the load data set will be presented to the object dataset converter (CDC), which produces an executable module.

Note that the load data set is created as a data generation set of depth one. Each time the program is recompiled, the last data set is erased and replaced with the one currently being generated.

DECK or NODECK: Specifying DECK causes the compiler to put the object code, in card image form, into a data set called LOAD.xxx(0), where xxx is the object module name. This option should be considered in conjunction with the LOAD or NOLCAD option.

LISTING OPTIONS

Listing options specify the information to be included in the compiler listing.

LINECNT: This option specifies the number of lines to be included in each page of a printed listing, including heading lines and blank lines. Three decimal digits are used.

OPLIST or NOOPLIST: This option requests a list showing the status of all the compiler options at the start of compilation.

SOURCE2 or NOSOURCE2: Specify SOURCE2 if you want a listing of the PL/I source statements input to the preprocessor.

SOURCE or NOSOURCE: Specify SOURCE if you want a listing of the PL/I source statements processed by the compiler. The source statements listed are either those of the original source program or the output from the preprocessor.

NEST or NONEST: Specify NEST if you want the source program listing to indicate, for each statement, the block level and the level of nesting of a DO group.

ATR or NOATR: Specify ATR if you want included in the listing a table of source program identifiers and their attributes. Attributes with a precision of fixed binary (15.0) or less are flagged '*********'. An aggregate length table, giving the length in bytes of all major structures and nonstructured arrays in the source program, is also produced when the ATR option is specified.

XREF or NOXREF: Specify XREF if you want included in the listing a cross-reference table that lists all the identifiers in the source program with the numbers of the source statements in which they appear. If you specify ATR and XREF, the two tables are combined. An Aggregate Length Table is also produced when the XREF option is specified.

EXTREF or NOEXTREF: Specify EXTREF if you want a listing of the external symbol dictionary (ESD).

LIST or NOLIST: Specify LIST if you want the machine instructions generated by the compiler (in a form similar to OS, OS/VS assembler language instructions).

FLAGW or FLAGE or FLAGS: Messages are listed in order of their occurrence on the user's terminal and in order of their severity in the output listing. There are four classes of diagnostic messages, which are graded in order of severity:

A warning is a message that calls attention to a possible error, although the statement to which it refers is syntactically valid. In

addition to alerting you, it may assist in writing more efficient programs in the future.

An error message describes an attempt to correct an erroneous statement; you are informed of the correction. Errors do not normally terminate processing of the text.

A severe error message indicates an error that cannot be corrected by the compiler. The incorrect section of the program is deleted, but compilation is continued. Where reasonable, the ERROR condition is made known when the object module is executed, if execution of an incorrect source statement is attempted. If a severe error occurs during compilation, compilation is terminated after the SOURCE listing is produced.

A terminal error message describes an error that, when discovered, forces the termination of the compilation.

You can select the severity at and above which diagnostic messages appear on the output listing. The meaning of each option is as follows:

FLAGW -- list all diagnostic messages

FLAGE -- list all diagnostic messages except warning messages

FLAGS -- list only severe errors and termination errors

DUMMY OPTIONS

Dummy options are included solely to give compatability with the PL/I (F) Compiler.

Size: This option is used in the PL/I (F) compiler to specify the amount of main storage available. The option has no effect in TSS PL/I and is included to give compatability with PL/I. Since there is a standard default built into the compiler, you need never take account of this option.

M91 or NOM91: This option is used to indicate if the machine is a Model 91. It is included for the same reasons as given above for SIZE.

EXTDIC or NOEXTDIC: This option is ignored, since the TSS PL/I compiler always takes the EXTDIC option.

This appendix lists and defines the COBOL/VS Compiler Options.

SIZE=YYYYYYY
indicates the amount of main storage, in bytes, available for compilation.

BUF=YYYYYY
indicates the amount of main storage to be allocated to buffers. If both SIZE and BUF are specified, the amount allocated to buffers is included in the amount of main storage available for compilation.

Note: The SIZE and BUF compile-time parameters can be given in multiples of K, where K=1024 decimal bytes. For example, 131,072 decimal bytes can be specified as 128K.

SOURCE|NOSOURCE
indicates whether or not the source module is to be listed.

CLIST|NOLIST
indicates whether or not a condensed listing is to be produced. If specified, the procedure portion of the listing will contain generated card numbers (unless the NUM option is in effect), verb references, and the location of the first instruction generated for each verb. Global tables, literal pools, register assignments, and information about the working-storage section are also provided. CLIST and PMAP are mutually exclusive options.

Note: In nonsegmented programs, verbs are listed in source order. in segmented programs, the root segment is last. (For programs run with the OPTIMIZE option the root segment is first, followed by the individual segments in order of ascending priority.)

DMAP|NODMAP
indicates whether or not a glossary is to be listed. Global tables, literal pools, register assignments, and information about the working-storage section are also provided.

PMAP|NOPMAP
indicates whether or not register assignments, global tables, literal pools, information about the working-storage section, and an assembler-language expansion of the source modules are to be listed. CLIST and PMAP are mutually exclusive options.

Note: If any one of the options CLIST, DMAP, and PMAP is specified, the compiler will produce a message giving the hexadecimal length and starting address of the working-storage section.

VERB|NOVERB
indicates whether procedure-names and verb-names are to be listed with the associated code on the object-program listing. VERB has meaning only if PMAP or CLIST is in effect. NOVERB yields more efficient compilation.

LOAD|NOLOAD
indicates whether or not the object module is to be written to a dataset named PUNCH.module.

DECK|NODECK
indicates whether or not the object module is to be written to a LOAD.module. This option is required if the module is to be converted by the object deck converter to a TSS loadable module.

SEQ|NOSEQ
indicates whether or not the compiler is to check the sequence of the source module statements. If the statements are not in sequence, a message is printed.

LINECNT=nn
indicates the number of lines to be printed on each page of the compilation source card listing. The number specified by nn must be a 2-digit integer from 01 to 99. If the LINECNT option is omitted, 60 lines are printed on each page of the output listing.

Note: The compiler allows for headings three lines less than the user has specified. (For example, if nn=55 is specified, then 52 lines are printed on each page of the output listing.)

ZWB|NOZWB
indicates whether or not the compiler generates code to strip the sign from a signed external decimal field when comparing this field to an alphanumeric field. If ZWB is specified, the signed external decimal field is moved to an intermediate field, in which its sign is removed, before it is compared to the alphanumeric field. ZWB complies with the ANS standard; NOZWB should be used when, for example, input numeric fields are to be compared with SPACES.

LVL=A/B/C/D|NOLVL
specifies what level of FIPS (federal Information Processing Standard) flagging is to be used. If flagging is specified, source clauses and statements that do not conform to the specified level of FIPS are identified. See the publication IBM OS Full American National Standard COBOL for a complete list of the statements flagged at each level.

Note: If LVL is the default, its assigned value can be overridden at compile time with any level except NOLVL. If NOLVL is the SYSGEN default, it can be overridden at compile time with any level. If the LVL option is in effect, the SYSUT6 dataset must be specified. If both LVL=A/B/C/D and TERM are specified, the compiler listing output for options such as SOURCE, PMAP, and XREF are not produced.

FLAGW|FLAGE
indicates the type of messages that are to be listed for the compilation. FLAGW indicates that all warning and diagnostic messages are to be listed. FLAGE indicates that all diagnostic messages are to be listed, but that the warning messages are not to be listed.

SUPMAP|NOSUPMAP
indicates whether or not the object code listing, and object module and link edit decks are to be suppressed if an E-level or D-level message is generated by the compiler.

SPACE1|SPACE2|SPACE3
indicates the type of spacing that is to be used on the source card listing generated when SOURCE is specified. SPACE1 specifies single spacing, SPACE2 specifies double spacing, and SPACE3 specifies triple spacing.

TRUNC|NOTRUNC
applies to movement of COMPUTATIONAL arithmetic fields. If TRUNC (standard truncation) is specified and the number of digits in the sending field is greater than the number of digits in the receiving

field, the arithmetic item is truncated to the number of digits specified in the PICTURE clause of the receiving field when moved. if NOTRUNC is specified, movement of the item is dependent on the size of the field (halfword, fullword).

QUOTE|APOST
indicates to the compiler that either the double quote(") or the apostropne (') is acceptable as the character to delineate literals and to use that character in the generation of figurative constants.

STATE|NOSTATE
indicates whether or not the number of the COEOL statement being executed at the time of an abnormal termination is desired. STATE identifies the number of the statement and the number of the verb being executed.

SYMDMP|NOSYMDMP
requests a formatted dump of the data area of the object program at abnormal termination. With this option, the programmer may request dynamic dumps of specified data-names at strategic points during program execution.

Notes: If the SYMDMP option is in effect, the SYSUT5 data set must be filedeffed. If the BATCH option is requested, symbolic debugging is rejected. Specification of the SYMDMP option automatically yields the OPTIMIZE feature, discussed below, and rejects the STATE option because SYMDMP output includes STATE output at abnormal termination.

OPTIMIZE|NOOPTIMIZE
causes optimized object code to be generated by the compiler, considerably reducing the use of object program main storage. In general, the greater the number of COBOL Procedure Division source statements, the greater the percentage of reduction in the amount of main storage required.

Note The optimizer feature is automatically in effect when the SYMDMP feature is specified.

SYNTAX|NOSYNTAX
CSYNTAX|NOCSYNTAX
indicates whether the source text is to be scanned for syntax errors only and appropriate error message are to be generated. For conditional syntax checking (CSYNTAX), a full compilation is produced so long as no messages exceed the W or C level. If one or more e-level or higher severity messages are produced, the compiler generates the messages but does not generate object text.

Notes:

1. When the SYNTAX option is in effect, all of the following compile-time options are suppressed:

LOAD	DECK	NAME
XREF	SYMDMP	COUNT
SXREF	TRUNC	VBSUM
CLIST	OPTIMIZE	VBREF
NOSUPMAP	PMAP	STATE

2. Unconditional syntax checking is assumed if all of the following compile-time options are specified:

NOLOAD	NOCLIST	SUPMAP
NOXREF	NOPMAP	NODECK
NOSXREF		

3. CSYNTAX and SYNTAX are mutually exclusive. CSYNTAX will override SYNTAX.

NUM|NONUM
indicates whether or not line numbers have been recorded in the input and, rather than compiler-generated source numbers, should be used in error messages, as well as in PMAP, CLIST, STATE, XREF, SXREF, and FLOW. NONUM indicates that the compiler-generated numbers should be used in error messages as well as in PMAP, CLIST, STATE, XREF, and SXREF.

XREF|NOXREF
indicates whether or not a cross-reference listing is produced. If XREF is specified, an unsorted listing is produced with data-names and procedure-names appearing in two parts in source order.

SXREF|NOSXREF
indicates whether or not a sorted cross-reference listing is produced. if SXREF is specified, a sorted listing is produced with data-names and procedure-names in alphanumeric order.

Note: XREF and SXREF are mutually exclusive.

LIB|NOLIB
indicates whether or not a COPY and/or a BASIS request will be part of the COBOL source input stream. If no library facilities are to be used, the specification of NOLIB will save compilation time.

BATCH|NOBATCH
indicates whether or not multiple programs and/or subprograms are to be compiled with a single invocation of the compiler.

NAME|NONAME
indicates whether or not programs in a multiple compilation environment will be combined into one or more load modules. If NAME is specified, each succeeding program will be put into a separate load module. this option will remain in effect for the entire compilation unless NONAME is specified for an individual program. Names for the load modules will be formed according to the rules for forming module names from the PROGRAM-ID.

Note: If the BATCH option is not specified, NONAME will be in effect.

RESIDENT|NORESIDENT
requests the COBOL Library Management feature. When one program in a given region/partition requests the RESIDENT option, the main program and all subprograms in that retion/partition should also request it.

Note: The RESIDENT option is automatically in effect when the DYNAM option is invoked.

DYNAM|NODYNAM
causes subprograms invoked through the CALL literal statement to be dynamically loaded and through the CANCEL statement to be dynamically deleted at object time (instead of link-edited with the calling program into a single load module).

Note: When both NORESIDENT and NODYNAM are either specified or implied by default, and a CALL identifier statement occurs in the source statement being compiled, the COBOL Library Managemen FACILITY OPTION (RESIDENT) is automatically in effect. A printed statement of this is given in the compiler output.

SYST|SYSx
indicates whether SYSOUT of SYSOUx, where x must be alphanumeric (that is, 0-9 or a-z except for t), is the ddname of the file to be used for debug output (READY TRACE, EXHIBIT) or DISPLAY statement. The specification in the program that is first to access the file is chosen.

ENDJOB|NOENDJOB
indicates whether or not, at the end of each job, COBOL library subroutines are to be called to delete modules, free main storage acquired through GETMAINs issued by the COBOL program or COBCL library subroutines, close DCBs opened by subroutines, and free their associated buffers. Specifying ENDJOB prevents fragmentation of main storage for programs executed on the system after the COBOL program. This option takes effect at a STOP RUN statement in any program and at a GOBACK statement in a main program only.

ADV|NOADV
indicates whether or not records for files with WRITE...ADVANCING need reserve the first byte for the control character. ADV specifies that the first byte need not be reserved.

COUNT|NOCOUNT
indicates whether or not code is to be generated to produce verb execution summaries at the end of problem program execution. Each verb is identified by procedure-name and by statement number, and the number of times it was used is indicated. In addition, the percentage of verb execution for each verb with respect to the execution of all verbs is given. A summary of all executable verbs used in a program and the number of times they are executed is provided. COUNT implies VERB.

DUMP|NODUMP
specifies that an abnormal termination dump is to be produced in the event of certain disaster level errors during compilation. NODUMP specifies that only an error message is to be produced in the event of these disaster level errors.
The dump produced will contain a four digit user completion code. Have the following available before calling IBM for programming support: source deck, control cards, and compiler output.

VBSUM|NOVBSUM
provides a brief summary of verbs used in the program and a count of how often each verb was used. This option provides the user with a quick search for specific types of statements. VBSUM implies VERB.

VBREF|NOVBREF
provides a cross-reference of all verbs used in the program. This option provides the programmer with a quick index to any verb used in the program. VBREF implies VERB and VBSUM.

Options for the Lister Feature

There are five compiler options for using the lister feature of the compiler. Note that either LSTONLY or LSTCOMP must be selected for the other four options to have meaning unless the BATCH option is specified.

LSTONLY|LSTCOMP|NOLST
indicates whether the lister feature is to be used. LSTONLY specifies that a reformatted listing is to be produced but that no compilation is to occur. LSTCOMP specifies that both a reformatted listing is to be produced and compilation is to occur.

FDECK|NOFDECK
indicates whether a copy of the reformatted source program is to be written on the SYSPUNCH data set. Since LDECK has meaning only with either LSTONLY or LSTCOMP, the lister output will be both a reformatted listing and a reformatted deck.

CDECK|NOCDECK
indicates whether or not COPY statements are to be expanded into COPY members in the SYSPUNCH output. The COPY members are to be expanded in the reformatted deck requested through FDECK. if CDECK is specified with NOFDECK, only the expanded COPY statements are produced.

LCCL1|LCCL2
indicates whether the Procedure Division part of the listing is to be in single or double column format.

L120|L132
indicates whether the length of each line of the reformatted listing is to be 120 or 132 characters long.

PRINT{(*|dsname)}|NOPRINT
indicates whether or not the program listing is to be suppressed, placed on the output data set specified by dsname, or displayed at the terminal. if PRINT is specified, the listing will include page headings, line numbers of the statements in error, message identification numbers, severity levels, and message texts (as well as any other output requested by SOURCE, CLIST, DMAP, PMAP, XREF, or SXREF). If (*) is specified instead of data-set name, the printed output is sent to the terminal. if PRINT alone is specified, a listing data set is created on secondary storage and named according to standard dataset naming conventions. NOPRINT specifies that no listing is to be printed. if neither PRINT nor NOPRINT is specified and any one or more of the options SOURCE, CLIST, DMAP, XREF, or PMAP are specified, PRINT is the default. Otherwise, NOPRINT is the default.

TERM|NOTERM
indicates whether or not progress and diagnostic messages are to be printed on the SYSTERM terminal data set. The severity level of the messages may be controlled by the FLAG option. If PRINT (*) is specified, then NOTERM is the default, to ensure that messages appear only once.

Default options are indicated by an underscore and need never be specified explicitly. The default options shown are standard IBM defaults; when the compiler is installed, each installation may establish its own set of default options.

Options may be coded in any order and may be separated by blanks or commas. As many as 100 characters may be coded.

SOURCE|NOSOURCE
indicates whether the source module listing is to be written. If SOURCE is specified, the source listing is produced in the listing dataset.

LINECOUNT(number)
indicates the maximum number of lines to be assigned per page of the source listing. The number may be in the range 1 to 99. If the option is omitted, the compiler assumes 60.

LIST|NOLIST
indicates whether the object module listing is to be written. The object module listing consists of statements written in psuedo-assembler language format.

OBJECT|NOOBJECT
indicates whether the object module (not the listing) is to be written.

DECK|NODECK
indicates whether the object module in card image form is to be produced. DECK is required to supply input to the object deck converter to produce a TSS loadable module.

OPTIMIZE({0|1|2})|NOOPTIMIZE
indicates what optimizing level is to be in force. NOOPTIMIZE indicates that no optimization is to be performed, and is equivalent to the specification OPTIMIZE(0). OPTIMIZE(1) specifies that each source module is to be treated by the compiler as a single program loop and that the single loop is to be optimized with regard to register allocation and branching. OPTIMIZE(2) specifies that each source module is to be treated as a collection of program loops and that each loop is to be optimized with regard to register allocation, branching, common expression elimination, and replacement of redundant computations.

FORMAT|NOFORMAT
indicates whether a structured source module listing is to be written. A structured source module listing indicates loop structures and the logical continuity of a source program. This option is effective only when OPTIMIZE(2) is in effect.

GOSTMT|NOGOSTMT
indicates whether internal sequence numbers (ISN) are to be generated for the calling sequence to subroutines for a traceback map. (a traceback map is a tool used in diagnosing execution errors).

MAP|NOMAP
indicates whether a table of names and statement labels used by the source program is to be written to the listing dataset.

XREF|NOXREF
indicates whether a cross reference listing of variables and labels used in the source program is to be written tc the listing dataset. If XREF is specified, ISNs are generated for each statement in which a variable or label is used.

NAME(name)
indicates the name to be given to the main source program. The name may be from one to six characters. If NAME is not specified, the compiler assumes the name MAIN.

BCD|EBCDIC
indicates whether the source module is written in BCD (Binary Coded Decimal) or EBCDIC (Extended Binary Coded Decimal Interchange Code). If BCD and EBCDIC statements are intermixed in the source module, BCD should be specified. BCD characters are not supported by the compiler as print control characters or in literal data. For example, the carriage control character to specify same line printing,+, is specified as a 12-8-6 punch in EBCDIC and as a 12 punch in BCD: the compiler recognizes only the EBCDIC code. Therefore, programs keypunched in BCD should be carefully screened for potential errors before job submission.

SIZE(MAX|nnnnk)
indicates the amount of main storage to be allocated to the compilation step. The symbol nnnnK represents the number, multiplied by K (1024-bytes), to be allocated. The number may range from 160 to 9999.

If SIZE(MAX) is specified, or if the option is omitted, the compiler uses all available storage in the environment in which it is operating.

AUTODBL(value)
calls the Automatic Precision Increase (API) facility and indicates whether data items are to be converted to higher precision. API provides an automatic means of converting single precision floating point calculations to double precision accuracy and double precision calculations to extended precision accuracy. the AUTODBL option indicates which particular data types are to be converted.

If AUTODBL is omitted, no precision increase is performed.

ALC|NOALC
indicates whether data items are to be aligned on proper storage boundaries. It is often used with the AUTODBL option to restore proper storage boundaries when a conversion is performed.

ANSF|NOANSF
indicates whether the compiler is to recognize only those library and built-in functions specified by the American National Standards Institute, (ANS), or the entire range of functions specified by IBM in the publication IBM FORTRAN IV LANGUAGE, Form No. GC28-6515. If ANSF is specified, any function not supported by ANS is considered to be user-supplied.

FLAG(I)|FLAGE(E)|FLAG(S)
indicates the level of diagnostic messages to be printed. FLAG(I) indicates that information messages, warning messages (those generating a return code of 4), error messages (those generating a return code of 8), and severe error messages (generating a return code of 12 or higher), are to be printed. FLAG(E) indicates that only error messages and severe error messages are to be printed. FLAG(S) indicates that only severe error messages are to be printed.

DUMP|NODUMP
indicates whether the contents of registers, storage, and files associated with the compiler are to be printed if an abnormal termination occurs. If DUMP is specified, a filedef statement named SYSUDUMP or SYSABEND must be supplied.

Changing Program Options During Compilation

The programmer may compile more than one source module. To change the options for any source module, the programmer precedes the source module with a card containing the characters *PROCESS in columns 1 through 9 followed by the options up to column 72, which must be left blank and which denotes the end of the *PROCESS card.

An example of the *PROCESS card is:

```
*PROCESS LIST,MAP
```

If succeeding source modules are not preceded by a *PROCESS card, options revert to those specified in the EXEC statement. Any option except SIZE may be specified on the *PROCESS card.

APPENDIX L: PL/I OPTIMIZING COMPILER OPTIONS

The compiler options are of the following types:

1. Simple pairs of keywords: a positive form (for example, NEST) that requests a facility, and an alternative negative form (for example, NONEST) that rejects that facility.

2. Keywords that permit you to provide a value-list that qualifies the option (for example, NOCOMPILE(E)).

3. A combination of 1 and 2 above.

The following paragraphs describe the options in alphabetic order.

AGGREGATE Option

The AGGREGATE option specifies that the compiler is to include in the compiler listing an aggregate length table, giving the lengths of all arrays and major structures in the source program.

ATTRIBUTES Option

The ATTRIBUTES option specifies that the compiler is to include in the compiler listing a table of source-program identifiers and their attributes. If both ATTRIBUTES and XREF apply, the two tables are combined.

CHARSET Option

The CHARSET option specifies the character set and data code that you have used to create the source program. The compiler will accept source programs written in the 60-character set or the 48-character set, and in the Extended Binary Coded Decimal Interchange Code(EBCDIC or Binary Coded Decimal (BCD).

60- or 48- character Set: If the source program is written in the 60-character set, specify CHARSET (60); if it is written in the 48-character set, specify CHARSET (48).

The language reference manual for this compiler lists both of these character sets. (The compiler will accept source programs written in either character set if CHARSET (48) is specified).

BCD or EBCDIC: If the source program is written in BCD, specify CHARSET (BCD); if it is written in EBCDIC, specify CHARSET (EBCDIC). The language reference manual for this compiler lists the EBCDIC representation of both the 48-character set and the 60-character set.

If both arguments (48 or 60, EBCDIC or BCD) are specified, they may be in any order and should be separated by a blank or by a comma.

COMPILE Option

The COMPILE option specifies that the compiler is to compile the source program unless an unrecoverable error was detected during preprocessing or syntax checking. The NOCOMPILE option without an argument causes processing to stop unconditionally after syntax checking. With an argument, continuation depends on the severity of errors detected so far, as follows

NOCOMPILE(W) ---- No compilation if a warning, error, sever error or unrecoverable error is detected.

NOCOMPILE(E) ---- No compilation if error, severe error,or unrecoverable error is detected.

NOCOMPILE(S) --- No compilation if a severe error or unrecoverable error is detected.

If the compilation is terminated by the NOCOMPILE option, the cross-reference listing and attribute listing may be produced; the other listings that follow the source program will not be produced.

CONTROL Option

The CONTROL option specifies that any compiler options deleted for your installation are to be available for this compilation. You must still specify the appropriate keywords to use the options. The CONTROL option must be specified with a password that is established for each installation; use of an incorrect password will cause processing to be terminated. the CONTROL option, if used, must be specified first in the list of options. It has the format:

```
CONTROL('password')
```

where "password" is a character string, not exceeding eight characters.

COUNT Option

The COUNT option specifies that the compiler i s to produce code to enable the number of times each statement is executed to b e counted at execution time.

The COUNT option implies the GOSTMT option if the STMT option applies, or the GONUMBER option if the NUMBER option applies.

DECK Option

The DECK option specifies that the compiler is to produce an object module in the form of 80-column card images and store it in the data set defined by the DD statement with the name SYSPUNCH. Columns 73-76 of each card contain a code to identify the object module; this code comprises the first four characters of the first label in the external procedure represented by the object module. Columns 77-80 contain a 4-digit decimal numberth e first card is numbered 0001, the second 0002, and so on. This option is req uired to produce a dataset to be converted to a TSS loadable module.

DUMP Option

The DUMP option specifies that the compiler is to produce a formatted dump of main storage if the compilation terminates abnormally (usually due to an I/O error or compiler error). This dump is written on the data set associated with SYSPRINT.

ESD Option

The ESD option specifies that the external symbol dictionary (ESD) is to be listed in the compiler listing.

FLAG Option

The FLAG option specifies the minimum severity of error that requires a message to be listed in the compiler listing:

FLAG(I) --- List all messages.

FLAG(W) --- List all except informatory messages. If you specify FLAG, FLAG(W) is assumed.

FLAG(E) --- List all except warning and informatory messages.

FLAG(S) --- List only severe error and unrecoverable error messages.

GONUMBER Option

The GONUMBER option specifies that the compiler is to produce additional information that will allow line numbers from the source program to be included in execution-time messages. Alternatively, these line numbers can be derived by using the offset address, which is always included in execution-time messages, and the table produced by the OFFSET option. (the NUMBER option must also apply.)

Use of the GONUMBER option implies NUMBER, NOSTMT, and NOGOSTMT.

GOSTMT Option

The GOSTMT option specifies that the compiler is to produce additional information that will allow statement numbers from the source program to be included in execution-time messages. Alternatively, these statement numbers can be derived by using the offset address, which is always included in execution-time messages, and the table produced by the OFFSET option. (the STMT option must also apply.)

Use of the GOSTMT, NOGONUMBER option implies STMT and nonumber.

INCLUDE Option

The INCLUDE option requests the compiler to handle the inclusion of PL/I source statements for programs that use the %INCLUDE statement. For programs that use the %INCLUDE statement but no other PL/1 preprocessor statements, this method is faster than using the preprocessor. If the MACRO option is also specified, the INCLUDE option has no effect.

INSOURCE Option

The INSOURCE option specifies that the compiler is to include a listing of the source program (including preprocessor statements) in the compiler listing. This option is applicable only when the preprocessor is used, therefore the MACRO option must also apply.

LINECOUNT Option

The LINECOUNT option specifies the number of lines to be included in each page of the compiler listing, including heading lines and blank lines. the format of the LINECOUNT option is:

LINECOUNT(n)

where "n" is the number of lines. It must be in the range 1 through 32767, but only headings are generated if you specify less than 7.

LIST Option

The LIST option specifies that the compiler is to include a listing of the object module (in a form similar to IBM assembler language instructions) in the compiler listing. The format of the list option is:

LIST[(m[,n])]

where "m" is the number of the first source statement for which an object listing is required and "n" is the number of the last source statement for which an object listing is required. If "n" is omitted, only statement "m" is listed. If the option NUMBER applies, "m" and "n" must be specified as line numbers.

LMESSAGE Option

The LMESSAGE and SMESSAGE options specify that the compiler is to produce messages in a long form (specify LMESSAGE) or in a short form (specify SMESSAGE).

MACRO Option

The MACRO option specifies that the source program is to be processed by the preprocessor.

MAP Option

The MAP option specifies that the compiler is to produce tables showing the organization of the static storage for the object module. These tables consist of a static internal storage map and the static external control sections. The MAP option is normally used with the LIST option.

MARGINI Option

The MARGINI option specifies that the compiler is to include a specified character in the column preceding the left-hand margin, and in the column following the right-hand margin of the listings resulting from the INSOURCE and SOURCE options. Any text in the source input which precedes the left-hand margin will be shifted left one column, and any text that follows the right-hand margin will be shifted right one column. For variable-length input records that do not extend as far as the right-hand margin, the character is inserted in the column following the end of the record. Thus text outside the source margins can be easily detected.

the MARGINI option has the format:

```
MARGINI('c')
```

where "c" is the character to be printed as the margin indicator.

MARGINS Option

The MARGINS option specifies the extent of the part of each input line or record that contains PL/I statements. The compiler will not process data that is outside these limits (but it will include it in the source listings).

The option can also specify the position of an American National Standard (ANS) printer control character to format the listing produced if the SOURCE option applies. This is an alternative to using %PAGE and %SKIP statements (described in the language reference manual for this compiler). If you do not use either method, the input records will be listed without any intervening blank lines. the format of the MARGINS option is:

```
MARGINS(m,n[c])
```

where "m" is the column number of the left-hand margin. It should not exceed 100.

"n" is the column number of the right-hand margin. It should be greater than m, but not greater than 100.

"c" is the column number of the ANS printer control characters. It should not exceed 100 and should be outside the values specified for m and n. Only the following control characters can be used:

(blank) skip one line before printing.
0 skip two lines before printing.
- skip three lines before printing.
+ no skip before printing.
1 start new page.

The standard IBM-supplied default for fixed-length records is MARGINS (2,72,0); that for variable-length and undefined-length records is MARGINS (10,100,0). A zero value for "c" specifies that there is no printer control character.

MDECK Option

The MDECK option specifies that the preprocessor is to produce a copy of its output (see MACRO option) and store it in the data set defined by SYSPUNCH, the load.module dataset. The last four bytes of each record in SYSUT1 are not copied, thus this option allows you to retain the output from the preprocessor as a deck of 80-column punched cards.

NAME Option

The NAME option specifies that the compiler is to place a NAME statement as the last statement of the object module. When processed by the object deck converter, this NAME statement indicates that primary input is complete and causes the specified name to be assigned to the load module created from the preceding input (since the last NAME statement).

It is required if you want the object deck converter to create more than one load module from the object modules produced by batched compilation.

If you do not use this option, the object deck converter will use the module name specified in the command. the format of the NAME option is:

```
NAME('name')
```

where name has from one through eight characters, and begins with an alphabetic character.

NEST Option

the NEST option specifies that the listing resulting from the SOURCE option will indicate, for each statement, the block level and the do-group level.

NUMBER option

The NUMBER option specifies that the numbers specified in the sequence fields in the source input records are to be used to derive the statement numbers in the listings resulting from the AGGREGATE, ATTRIBUTES, LIST, OFFSET, SOURCE and XREF options.

If NONUMBER is specified, STMT and NOGONUMBER are implied. NUMBER is implied by NOSTMT or GONUMBER.

The position of the sequence field can be specified in the SEQUENCE option. Alternatively the following default positions are assumed:

First 8 columns for undefined-length or variable-length source input records. In this case, 8 is added to the values used in the MARGINS option.

Last 8 columns for fixed-length source input records.

Note: The preprocessor output has fixed-length records irrespective of the original primary input. Any sequence numbers in the primary input are repositioned in columns 73-80.

The line number is calculated from the five right-hand characters of the sequence number (or the number specified, if less than five). These characters are converted to decimal digits if necessary. Each time a sequence number is found that is not greater than the preceding line

number, a new line number is formed by adding the minimum integral multiple of 100,000 necessary to produce a line number that is greater than the preceding one. If the sequence field consists only of blanks, the new line number is formed by adding 10 to the preceding one. The maximum line number permitted by the compiler is 134,000,000; numbers that would normally exceed this are set to this maximum value. Only eight digits are printed in the source listing; line numbers of 100,000,000 or over will be printed without the leading "1" digit.

If there is more than one statement on a line, a suffix is used to identify the actual statement in the messages. For example, the second statement beginning on the line numbered 40 will be identified by the number 40.2. The maximum value for this suffix is 31. Thus the thirty-first and subsequent statements on a line have the same number.

OBJECT Option

The OBJECT option specifies that the compiler is to store the object module that it creates in the data set defined by the ddname SYSLIN called punch.module.

OFFSET Option

The OFFSET option specifies that the compiler is to print a table of statement or line numbers for each procedure with their offset addresses relative to the primary entry point of the procedure. This information is of use in identifying the statement being executed when an error occurs and a listing of the object module (obtained by using the LIST option) is available. if GOSTMT applies, statement numbers, as well as offset addresses, will be included in execution-time messages. if GONUMBER applies, line numbers, as well as offset addresses, will be included in execution-time messages.

OPTIMIZE Option

The OPTIMIZE option specifies the type of optimization required:

NOOPTIMIZE -- specifies fast compilation speed, but inhibits optimization for faster execution and reduced main storage requirements.

OPTIMIZE (TIME) -- specifies that the compiler is to optimize the machine instructions generated to produce a very efficient object program. A secondary effect of this type of optimization can be a reduction in the amount of main storage required for the object module. The use of CPTIMIZE (TIME) could result in a substantial increase in compile time over NCOPTIMIZE.

OPTIMIZE(0) -- is the equivalent of NOOPTIMIZE.

OPTIMIZE(2) -- IS THE EQUIVALENT OF OPTIMIZE (TIME).

The language reference manual for this compiler includes a full discus sion of optimization.

OPTIONS Option

The OPTIONS option specifies that the compiler is to include in the compiler listing, a list showing the compiler options, to be used during this compilation.

SEQUENCE Option

The SEQUENCE option specifies the extent of the part of each input line or record that contains a sequence number. This number is included in the source listings produced by the INSOURCE and SOURCE option. Also, if the NUMBER option applies, line numbers will be derived from these sequence numbers and will be included in the source listing in place of statement numbers. No attempt is made

to sort the input lines or records into the specified sequence. The SEQUENCE option has the format:

SEQUENCE(m,n)

where "m" specifies the column number of the left-hand margin.
where "n" specifies the column number of the right-hand margin.

The extent speified should not overlap with the source program (as spe cified in the MARGINS option).

SIZE Option

This option can be used to limit the amount of main storage used by the ccmpiler. This is of value, for example, when dynamically invoking the compiler, to ensure that space is left for other purposes. The SIZE option can be expressed in five forms:

SIZE(YYYYYYYY) -- specifies that yyyyyyyy bytes of main storage are to be requested. leading zeros are not required.

SIZE(YYYYYK) -- specifies that YYYYYK bytes of main storage are to be requested (1K=1024). Leading zeros are not required.

SIZE(-YYYYYY) -- specifies that the compiler is to obtain as much main storage as it can, and then release YYYYYY bytes to the operating system. Leading zeros are not required.

SIZE(-YYYK) -- specifies that the compiler is to obtain as much main storage as it can, and then release YYK bytes to the operating system (1K=1024). Leading zeros are not required.

SIZE(MAX) -- specifies that the compiler is to obtain as much main storage as it can.

The IBM default, and the most usual value to be used is SIZE (MAX), which permits the compiler to use as much main storage in the partition or region as it can.

When a limit is specified, the amount of main storage used by the comp ilerdepends on how the operating system has been generated, and the method used for storage allocation.The ccmpiler assumes that buffers, data management routines, and processing phases take up a fixed amount of main storage, but this amount can vary unknown to the compiler.

MESSAGE Option

See LMESSAGE option.

SOURCE Option

The SOURCE option specifies that the compiler is to include in the compiler listing a listing of the source program. The source program listed is either the original source input or, if the MACRC option applies, the output from the preprocessor.

STMT Option

The STMT option specifies that statements in the source program are to be counted, and that this "statement number" is used to identify statements in the compiler listings resulting from the AGGREGATE, ATTRIBUTES, LIST, OFFSET, SOURCE, and XREF options. STMT is implied by NONUMBER or GOSTMT. If NOSTMT is specified, NUMBER and NOGOSTMT are implied.

STORAGE Option

The STORAGE option specifies that the compiler is to include in

thecompiler listing a table giving the main storage requirements for the object module.

SYNTAX Option
The SYNTAX option specifies that the compiler is to continue into syntax checking after initialization (or after preprocessing if the MACRO option applies) unless an unrecoverable error is detected.

The NOSYNTAX option without an argument causes processing to stop unconditionally after initialization (or preprocessing). With an argument, continuation depends on the severity of errors detected so far, as follows:

NOSYNTAX(W) -- No syntax checking if a warning, error, severe error, or unrecoverable error is detected.

NOSYNTAX(E) -- No syntax checking if an error, severe error, or unrecoverable error is detected.

NOSYNTAX(S) -- No syntax checking if a severe error or unrecoverable error is detected.

If the SOURCE option applies, the compiler will generate a source listing even if syntax checking is not performed.

If the compilation is terminated by the NOSYNTAX option, the cross-reference listing, attribute listing, and other listings that follow the source program will not be produced.

The use of this option can prevent wasted runs when debuggin a PL/I program that uses the preprocessor.

TERMINAL Option
It specifies that some or all of the compiler listing produced during compilation is to be printed at the terminal. If TERMINAL is specified without an argument, diagnostic and informatory messages are printed at the terminal. You can add an argument, which takes the form of an option list, to specify other parts of the compiler listing that are to be printed at the terminal.

The listing at the terminal is independent of that written on SYSPRINT.the following option keywords, their negative forms, or their abbreviated forms, can be specified in the option list:

AGGREGATE, ATTRIBUTES, ESD, INSOURCE, LIST, MAP, OPTIONS, SOURCE, STORAGE, and XREF.

The other options that relate to the listing (that is, FLAG, GONUMBER,GOSTMT, LINECOUNT, LMESSAGE/SMESSAGE, MARGINI, NEST, and NUMBER) will be the same as for the SYSPRINT listing.

XREF Option
The XREF option specifies that the compiler is to include in the compiler listing a list of all identifiers used in the PL/I program together with the numbers of the statements in which they are declared or referenced. Note that label references on END statements are not included, reference lists for structures may be incomplete, and arrays of structures are always listed with bounds of (*). If both ATTRIBUTES and XREF apply, the two tables are combined.

SPECIFYING EXECUTION-TIME OPTIONS

For each execution, the IBM or installation default for an execution-time option will apply unless it is overridden by a PLIXOPT string in the source program or by the PARM parameter of the OSRUN statement for execution.

An option specified in the PLIXOPT string overrides the default value, and an option specified in the PARM parameter overrides that specified in the PLIXOPT string.

Specifying Execution-time Options in the PLIXOPT String

Execution-time options can be specified in a source program by means of the following declaration:

```
DCL PLIXOPT CHAR (len) VAR INIT ('strng') STATIC EXTERNAL;
```

where "strng" is a list of options separated by commas, and "len" is a constant equal to or greater than the length of "strng".

If more than one external procedure in a job declares PLIXOPT as STATIC EXTERNAL, only the first string will be link-edited and available at execution time.

Specifying Execution-time Cptions in the OSRUN Command

You can also use the PARM field to pass an argument to the PL/I main procedure. To do so, place the argument, preceded by a slash, after the execution-time options. for example:

```
OSRUN OPT, 'ISASIZE (10K), REPORT/ARGUMENT
```

If you wish to pass an argument without specifying options, it must be preceded by a slash. for example:

```
OSRUN OPT, PARM='/ARGUMENT'
```

EXECUTION-TIME OPTIONS

The following paragraphs describe the execution-time options, which can be specified in the EXEC statement or in the PLIXOPT string.

COUNT
: specifies that a count is to be kept of the number of times each statement in the program is executed and that the results are to be printed when the program terminates.

NOCOUNT
: specifies that statement counting is not to be performed.

REPORT
: specifies that a report of certain program management activity is to be printed. The report will be automatically output to a data-set with the ddname PLIDUMP or PL1DUMP on program termination. This includes, for example, the amount of storage that was specified in the ISASIZE option, the length of the initial storage area, and the amount of PL/I storage required. This option may be abbreviated to R. The use of the report is described in "execution-time Storage Requirements", below.

NOREPORT
: specifies that a report is not required. This option may be abbreviated to NR.

STAE
specifies that when an ABEND SVC, but not a TSS abend, occurs, the PL/I library routines are to attempt to raise the ERROR condition or to produce a diagnostic message and a PLIDUMP.

NOSTAE
specifies that on program initialization, a STAE macro instruction is not to be issued.

SPIE
specifies that when a program interrupt occurs, the PL/I error handler is to be invoked. Under certain circumstances the ERROR condition will be raised.

NOSPIE
specifies that on program initialization, a SPIE macro instruction is not to be issued. This option must not be used if extended precision variables are used in the PL/I source program.

The execution-time options are discussed in greater detail in the publication OS PL/I Optimizing Compiler: Execution Logic.

INDEX

When more than one page reference is given, the major reference is first. All references are within plus or minus one of the indicated page numbers.

GC28-2001-9

IBM Time Sharing System Command System User's Guide Printed in U.S.A. GC28-2001-9

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
366 Hamilton Avenue, White Plains, New York 10601
(International)