File No. S360-36
Order No. GC28-6732-3

OS

IBM Systems Reference Library

IBM System/360 Operating System:

Time Sharing Option

Command Language Reference

OS Release 21

This reference publication describes the TSO command language that a terminal user may use to request the services of TSO.

The "Introduction" describes what the command language is. The section entitled "What You Must Know to Code the Commands" contains general information necessary for the use of every command.

The section entitled "The Commands" contains a description of each command, its operands and its subcommands. Examples are included.

"Command Procedure Statements" describes the statements designed for use in command procedures.

Fourth Edition (July, 1972)

This is a reprint of GC28-6732-3 incorporating changes released in the following Technical Newsletters:

GN28-2521 (dated April 15, 1972)
GN28-2531 (dated May 15, 1972)

This edition applies to release 21 as updated by component release 360S-OS-586, of IBM System/360 Operating System, and to all subsequent releases until otherwise indicated in new editions or Technical Newsletters. Changes are continually made to the information herein; before using this publication in connection with the operation of IBM Systems, consult the latest IBM System/360 and System/370 Bibliography, Order No. GA22-6822, and the current SRL Newsletter, Order No. GN20-0360, for the editions that are applicable and current.

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

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Publications Development, Department D58, Building 706-2, PO Box 390, Poughkeepsie, N.Y. 12602. Comments become the property of IBM.

Preface

This publication describes the commands and operands of the TSO Command Language. It is intended for use at a terminal. The level of knowledge required for this publication depends upon the command being used. Most commands require little knowledge of TSO and of the Operating System; however, some commands require a greater knowledge of the system. As a general rule, the description of each command requires an understanding of those elements being manipulated by the command.

The prerequisite publication, IBM System/360 Operating System: Time Sharing Option, Terminal User's Guide, GC28-6763 describes what commands are used to perform the following functions:

- Start and end a terminal session.
- Enter and manipulate data.
- Program at the terminal.
- Test a program.
- Write and use Command Procedures.
- Control a system with TSO.

Once a user is familiar with the Terminal User's Guide, he or she can use this publication to code the TSO Commands.

The publication, IBM System/360 Operating System: Time Sharing Option, Terminals, GC28-6762 describes how to use the terminals supported by TSO.

The major divisions in this book are:

- Introduction
- What You Must Know to Code the Commands
- The Commands
- Command Procedure Statements
- Index

The Introduction describes what the command language is. The section entitled "What You Must Know to Code the Commands" contains general information necessary for the use of every command.

The section entitled "The Commands" contains a description of each command, its operands and its subcommands. Examples are included.

The commands are presented in alphabetical order. Subcommands are presented in alphabetical order following the command to which they apply. A boldface heading on each page identifies the information contained on the page. The boldface headings and alphabetical organization allow you to locate particular commands as you would locate a subject in a dictionary or encyclopedia. The larger boldface headings identify the first pages of the descriptions of commands.

"Command Procedure Statements" describes the statements designed for use in command procedures.

The "Index" contains the location (page number) where terms and subjects are discussed in the text.

Contents

Figures

Summary of Amendments for GC28-6732-2 as Updated by GN28-2521 and GN28-2531 Component Release 360S-OS-586

DYNAMIC SPECIFICATION OF DATA SET ATTRIBUTES (DCB Parameters)

- A new command, ATTRIB, was added. By using this command, a TSO user can build and store a list of data set attributes. These attributes can subsequently be assigned to a data set allocated dynamically.
- A new operand, USING (attribute-list-name), was added to the ALLOCATE command.
- A new operand, ATTRLIST (attribute-list-names), was added to the FREE command.
- Changes were made to the introductory information and index where applicable.

MISCELLANEOUS CHANGES

- Typographical and syntatical errors are corrected in the ATTRIB and FREE commands.

Summary of Amendments for GC28-6732-2 OS Release 21

STATUS DISPLAY

The SQA operand has been added to the DISPLAY subcommand of OPERATOR.

QUOTED STRING NOTATION

The ability to use single quotes as delimiters for a character string has been added to the CHANGE and FIND subcommands of EDIT.

TECHNICAL CORRECTIONS

Technical corrections have been made throughout this publication, as indicated by a vertical line to the left of each change. Editorial corrections and clarifications have been made as required.

GLOSSARY

Terms that were duplicated in the IBM Data Processing Glossary, GC20-1699, have been removed.

WHAT YOU MUST KNOW TO CODE THE COMMANDS

Removed the list of TSO informational messages. They are documented in the publication: IBM System/360 Operating System: Messages and Codes, GC28-6631.

PROGRAM PRODUCT COMMANDS

Moved the Program Product Commands into Appendix C. Added references to new Program Product Publications to Appendix A.

Summary of Amendments for GC28-6732-1 as Updated by GN28-2503 OS Release 20.6

MODIFY SUBCOMMAND OF OPERATOR
The keyword descriptions have been clarified.

Summary of Amendments for GC28-6732-1 as Updated by GN28-2480 OS Release 20.1

MESSAGES
Messages added and changed

FORT DATA SET TYPE KEYWORD
Keyword deleted

CTLX KEYWORD FOR PROFILE COMMAND AND SUBCOMMAND
Keyword added

EQUATE KEYWORD FOR GETMAIN SUBCOMMAND
Keyword added

COPY SUBCOMMAND FOR TEST COMMAND
Subcommand added

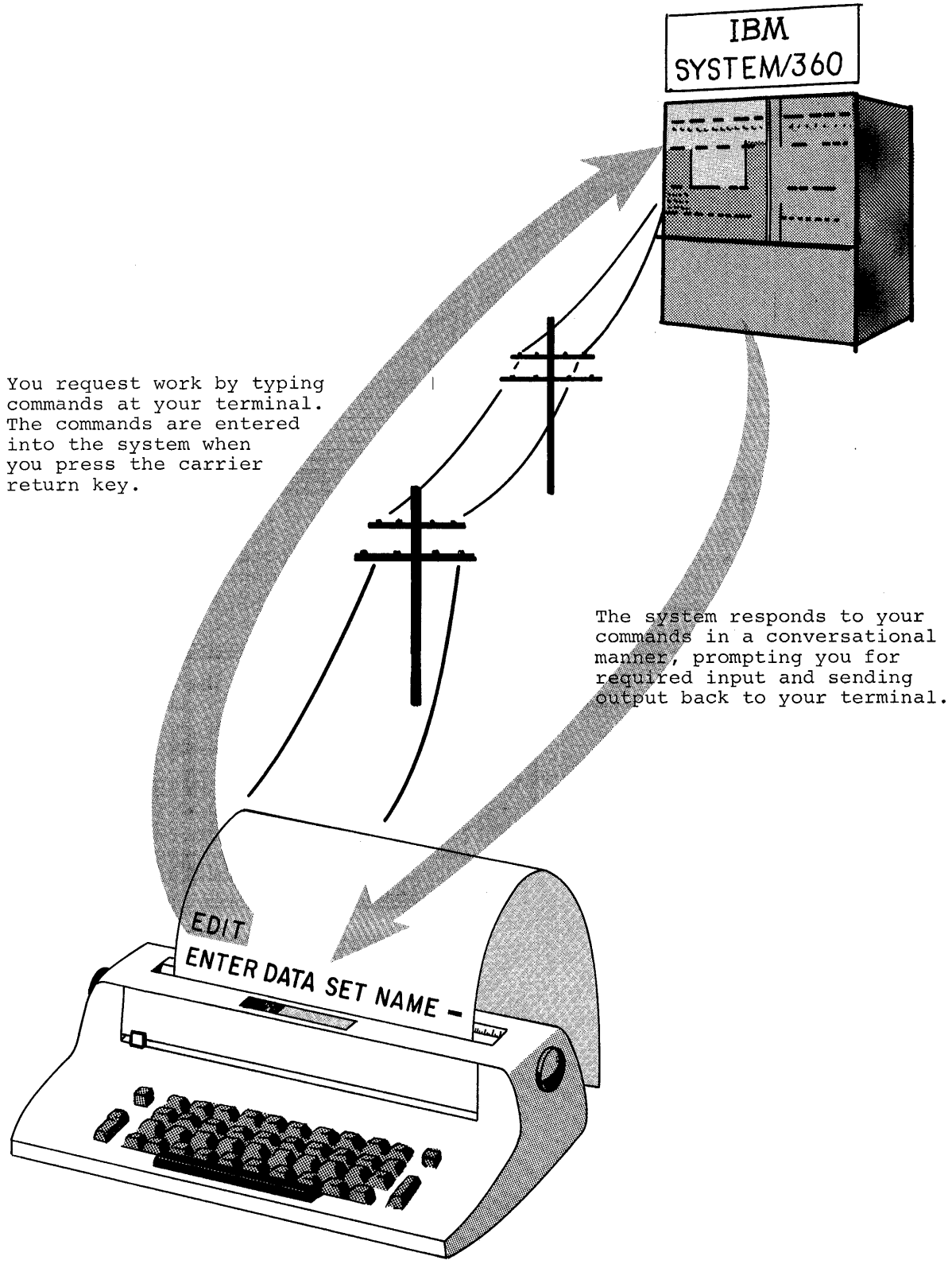


Figure 1. Entering Commands From a Terminal

Introduction

TSO is the Time Sharing Option of the System/360 Operating System. TSO allows you and a number of other users to use the facilities of the system concurrently and in a conversational manner. You can communicate with the system by typing requests for work (commands) on a terminal which may be located far away from the system installation. The system responds to your requests by performing the work and sending messages back to your terminal. The messages tell you such things as what the status of the system is with regard to your work and what input is needed to allow the work to be done.

A command, then, is a request for work. By using different commands, you can have different kinds of work performed. You can store data in the system, change the data, and retrieve it at your convenience. You can create programs, test them, have them executed, and obtain the results at your terminal. The commands make the full capability of the system available at your terminal.

When you use a command to request work, the command establishes the scope of the work to the system. To provide flexibility and greater ease of use, the scope of some commands' work encompasses several operations that are identified separately. After entering the command, you may specify one of the separately identified operations by typing a subcommand. A subcommand, like a command, is a request for work; however, the work requested by a subcommand is a particular operation within the scope of work established by a command.

The commands and subcommands recognized by TSO form the TSO command language. The command language is designed to be easy to use. The command names and subcommand names are typically familiar English words, usually verbs, that describe the work to be done. The number of command names and subcommand names that you must learn has been kept to a minimum. The information that you must provide is defined by operands (words or numbers that accompany the command names and subcommand names). Most of the operands have default values that are used by the system if you choose to omit the operand from the command or subcommand. In addition, you can abbreviate many of the command names, subcommand names and operands. Together, the defaults and abbreviations decrease the amount of typing required.

This reference manual describes what each command can do and how to enter, or type in, a command at your terminal. Figure 2 shows you the kinds of work you can accomplish by using the command language, and identifies most of the commands and subcommands that you can use to request each kind of work. A complete list of the commands, subcommands, and their abbreviations is located on the divider page that precedes the descriptions of the commands. The rules for abbreviating operands are in the section, "What You Must Know to Code the Commands."

Additional commands and subcommands are available for a license fee as optional Program Products. Appendix A contains information on the Program Products, along with references to related publications. Appendix C lists the Program Product commands and subcommands.

Figure 2. Functions of the TSO Commands and Subcommands (Part 1 of 2)

FUNCTION		COMMAND	SUBCOMMAND
CONTROL YOUR TERMINAL SESSION	identify yourself to the system.....	LOGON	
	define your operational characteristics..	TERMINAL	
		PROFILE	
		EDIT.....	PROFILE
	display messages (notices and mail).....	LISTBC	
	send messages.....	SEND	
	obtain help from the system.....	HELP	
		OPERATOR..	HELP
		ACCOUNT...	HELP
		EDIT.....	HELP
		OUTPUT....	HELP
		TEST.....	HELP
	end your terminal session.....	LOGOFF	
	display session time used.....	TIME	
ENTER, MODIFY, STORE, AND RETRIEVE DATA	create a data set.....	EDIT	
	enter data into a data set†.....	EDIT.....	INPUT
		EDIT.....	INSERT
	change data in a data set†.....	EDIT.....	CHANGE
	place data into columns.....	EDIT.....	TABSET
	change position of current line.....	EDIT.....	UP
		EDIT.....	DOWN
		EDIT.....	TOP
		EDIT.....	BOTTOM
		EDIT.....	FIND
	display referenced lines.....	EDIT.....	VERIFY
	renumber lines of data.....	EDIT.....	RENUM
	check the syntax of input statements.....	EDIT.....	SCAN
	delete lines of data from a data set†....	EDIT.....	DELETE
	delete an entire data set.....	DELETE	
	allocate a data set.....	ALLOCATE	
	specify attributes for a data set.....	ATTRIB	
	free an allocated data set or attributes.	FREE	
	copy a data set.....	COPY*	
	format a data set.....	FORMAT*	
		EDIT.....	FORMAT*
	merge two data sets.....	MERGE*	
		EDIT.....	MERGE*
	list the contents of a data set.....	LIST*	
		EDIT.....	LIST
	list the names of allocated data sets....	LISTALC	
	list the names of cataloged data sets....	LISTCAT	
	list information about your data sets....	LISTDS	
	store a data set.....	EDIT.....	SAVE
	rename a data set.....	RENAME	
	establish passwords for a data set.....	PROTECT	
	end the EDIT functions.....	EDIT.....	END

* optional Program Products, available for a license fee

† Insert/replace/delete function of EDIT can be used for single line operations.

Figure 2. Functions of the TSO Commands and Subcommands (Part 1 of 2)

Figure 2. Functions of the TSO Commands and Subcommands (Part 2 of 2)

FUNCTION		COMMAND	SUBCOMMAND
DEVELOP PROGRAMS AND PROCESS DATA	create a program; enter it in a data set.	EDIT	
	convert PL/1,BASIC, and FORTRAN statements	CONVERT	
	compile and execute a program	RUN	
		EDIT	RUN
	invoke a standard compiler	CALL	
	invoke the assembler prompter	ASM*	
	invoke the COBOL prompter	COBOL*	
	invoke the FORTRAN prompter	FORT*	
	linkage edit a compiled program	LINK	
	load and execute a load module	CALL LOADGO	
	load and execute an object module	LOADGO	
	submit a job for batch processing	SUBMIT	
	cancel a batch job	CANCEL	
	display the status of a batch job	STATUS OUTPUT	
	execute a procedure consisting of commands	EXEC	
TEST AND DEBUG A PROGRAM	initiate testing	TEST	
	load a load module for execution	TEST	LOAD
	set breakpoints for inspection	TEST	AT
	remove breakpoints	TEST	OFF
	establish base location for addresses	TEST	QUALIFY
	display absolute addresses	TEST	WHERE
	add symbols to symbol table	TEST	EQUATE
	initialize registers and start execution.	TEST	CALL
	copy data in main storage or in register locations	TEST	COPY
	start or restart program execution	TEST	GO
	get additional main storage	TEST	GETMAIN
	release main storage	TEST	FREEMAIN
	display contents of main storage	TEST	LIST
	display contents of registers	TEST	LIST
	display a Data Control Block	TEST	LISTDCB
	display a Data Extent Block	TEST	LISTDEB
	display a storage map	TEST	LISTMAP
	display the Program Status Word	TEST	LISTPSW
	display a Task Control Block	TEST	LISTTCB
	delete the program with no more testing	TEST	RUN
	end testing	TEST	END
	display CPU time	TIME	
CONTROL THE SYSTEM	modify TSO options	OPERATOR	MODIFY
	display system status	OPERATOR	DISPLAY
	cancel a terminal user	OPERATOR	CANCEL
	monitor terminal and job activities	OPERATOR	MONITOR
	send messages to users	OPERATOR	SEND
	stop monitoring	OPERATOR	STOPMN
	maintain the user attribute data set	ACCOUNT	
	add new user attributes	ACCOUNT	ADD
	delete user attributes	ACCOUNT	DELETE
	change user attributes	ACCOUNT	CHANGE
	display user attributes	ACCOUNT	LIST
	list user identifications	ACCOUNT	LISTIDS
* optional Program Products, available for a license fee.			

Figure 2. Functions of the TSO Commands and Subcommands (Part 2 of 2)

What You Must Know To Code The Commands

To use the TSO command language you should know:

- The syntax of a command.
- The way to enter a command.
- The data set naming conventions.

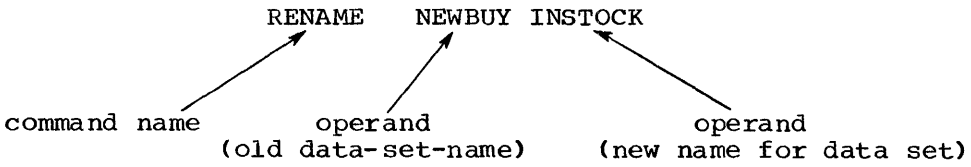
In addition, you should be aware of the aids available to you:

- The attention interruption.
- The HELP command.
- The messages that you receive from the system.

Note: In this manual, all references to terminal keyboards and keys apply specifically to the IBM 2741 Communications Terminal. For information concerning the use of other terminals refer to IBM System/360 Operating System: Time Sharing Option, Terminals, GC28-6762-0. Terminals which are equivalent to those explicitly supported may also function satisfactorily. The customer is responsible for establishing equivalency. IBM assumes no responsibility for the impact that any changes to the IBM-supplied products or programs may have on such terminals.

THE SYNTAX OF A COMMAND

A command consists of a command name followed, usually, by one or more operands. A command name is typically a familiar English word, usually a verb, that describes the function of the command. For instance, the RENAME command changes the name of a data set. Operands provide the specific information required for the command to perform the requested operation. For instance, operands for the RENAME command identify the data set to be renamed and specify the new name:



Two types of operands are used with the commands: positional and keyword. Positional operands follow the command name and precede keywords.

Positional Operands

Positional operands are values that follow the command name in a prescribed sequence. The value may be one or more names, symbols, or integers. In the command descriptions within this manual, the positional operands are shown in lower case characters. A typical positional operand is:

data-set-name

You must replace "data-set-name" with an actual data set name when you enter the command.

When you want to enter a positional operand that is a list of several names or values, the list must be enclosed within parentheses. The names or values must not include unmatched right parentheses.

Keyword Operands

Keywords are specific names or symbols that have a particular meaning to the system. You can include keywords in any order following the positional operands. In the command descriptions within this book, keywords are shown in upper case characters. A typical keyword is:

TEXT

In some cases you may specify values with a keyword. The value is entered within parentheses following the keyword. The way a typical keyword with a value appears in this book is:

LINESIZE(integer)

Continuing this example, you would select the number of characters that you want to appear in a line and substitute that number for the "integer" when you enter the operand:

LINESIZE(80)

Abbreviating Keyword Operands

You must enter keywords spelled exactly as they are shown or you may use an acceptable abbreviation. You may abbreviate any keyword by entering only the significant characters; that is, you must type as much of the keyword as is necessary to distinguish it from the other keywords of the command or subcommand. For instance, the LISTBC command has four keywords:

MAIL	NOTICES
NOMAIL	NONOTICES

The abbreviations are:

M for MAIL (also MA and MAI)
NOM for NOMAIL (also NOMA and NOMAI)
NOT for NOTICES (also NOTI, NOTIC, and NOTICE)
NON for NONOTICES (also NONO, NONOT, NONOTI, NONOTIC, and NONOTICE)

Delimiters

When you type a command, you should separate the command name from the first operand by one or more blanks. You should separate operands by one or more blanks or a comma. For instance, you can type the LISTBC command like this:

```
LISTBC NOMAIL NONOTICES
```

or like this:

```
LISTBC NOMAIL,NONOTICES
```

or like this: LISTBC NOMAIL NOTICES

Enter a blank by pressing the space bar at the bottom of your terminal keyboard. You can also use the TAB key to enter one or more blanks.

Note: A keyword with a value is a single operand and must not contain delimiters; for instance, do not separate the keyword from the parentheses that enclose the value.

Syntax Notation Conventions

The notation used to define the command syntax and format in this publication is described in the following paragraphs.

1. The set of symbols listed below is used to define the format but you should never type them in the actual statement.

hyphen	-
underscore	_
braces	{}
brackets	[]
ellipsis	...

   The special uses of these symbols are explained in paragraphs 5-9.

2. You should type upper-case letters and words, numbers, and the set of symbols listed below in an actual command exactly as shown in the statement definition.

apostrophe	'
asterisk	*
comma	,
equal sign	=
parentheses	()
period	.

3. Lower-case letters, words, and symbols appearing in a command definition represent variables for which you should substitute specific information in the actual command.

   Example: If name appears in a command definition, you should substitute a specific value (for example, ALPHA) for the variable when you enter the command.

4. Stacked items represent alternatives. You should select only one such alternative.

   Example: The representation

   A
   B
   C

   indicates that either A or B or C is to be selected.

5. Hyphens join lower-case letters, words, and symbols to form a single variable.

   Example: If member-name appears in a command definition, you should substitute a specific value (for example, BETA) for the variable in the actual command.

6. An underscore indicates a default option. If you select an underscored alternative, you need not type it when you enter the command.

   Example: The representation

   $$\underline{A}\ B\ C$$

   indicates that you are to select either A or B or C; however, if you select B, you need not type it, because it is the default option.

7. Braces group related items, such as alternatives.

   Example: The representation

   $$ALPHA=(\{A\ B\ C\},D)$$

   indicates that you must choose one of the items enclosed within the braces. If you select A, the result is ALPHA=(A,D).

8. Brackets also group related items; however, everything within the brackets is optional and may be omitted.

   Example: The representation

   $$ALPHA=(\left[\underline{A}\ B\ C\right],D)$$

   indicates that you may choose one of the items enclosed within the brackets or that you may omit all of the items within the brackets. If you select B, the result can be either ALPHA=(,D) or ALPHA=(B,D).

9. An ellipsis indicates that the preceding item or group of items can be repeated more than once in succession.

   Example:

   ```
   ALPHA[,BETA...]
   ```

   indicates that ALPHA can appear alone or can be followed by ,BETA any number of times in succession.

Subcommands

The work done by some of the commands is divided into individual operations. Each operation is defined and requested by a subcommand. To request one of the individual operations, you must first enter the command. You can then enter a subcommand to specify the particular operation that you want performed. You can continue entering subcommands until you enter the END subcommand.

The commands that have subcommands are ACCOUNT, CALC (a Program Product), EDIT, OPERATOR, OUTPUT and TEST. When you enter the ACCOUNT command you can then enter the subcommands for ACCOUNT. Likewise, when

you enter the CALC, EDIT, OPERATOR, OUTPUT, or TEST commands you can enter appropriate subcommands.

The syntax of a subcommand is the same as that of a command. A subcommand consists of a subcommand name followed, usually, by one or more operands. The discussions of operands and delimiters apply to subcommands as well as commands.

HOW TO ENTER A COMMAND

A terminal session is designed to be an uncomplicated process: you identify yourself to the system by entering the LOGON command and then request work from the system by entering other commands. To enter a command or subcommand:

1. Type the command or subcommand name and any operands that you select.
2. Press the carrier return key.

You can begin typing at any position on a line; you do not have to start at the lefthand margin. You can type command names and operands in either uppercase or lowercase characters. You may prefer to type your input in lowercase characters so that you can distinguish your input from the system's messages on your listing (the system prints in uppercase characters).

You can continue a line by placing a hyphen at the end of the line that is to be continued. For a discussion of data continuation under EDIT, see the topic, Modes of Operation under the EDIT command.

You can define your own character-deletion and line-deletion characters for correcting typing errors, or you can accept the characters that the system uses by default (if you do not specify your own). The default characters for the IBM 2741 Communications Terminal are:

- The BACKSPACE key-to delete the preceding character on the line.
- The ATTN key-to delete the entire line (including continued lines).

For other defaults and for information concerning the use of other terminals refer to IBM System/360 Operating System: Time Sharing Option, Terminals, GC28-6762.

You may use the PROFILE command to define the keys that you want to use as the character-deletion and line-deletion characters.

DATA SET NAMING CONVENTIONS

A data set is a collection of data. Each data set stored in the system is identified by a unique data set name. The data set name allows the data to be retrieved and helps protect the data from unauthorized use.

The data set naming conventions for TSO simplify the use of data set names. When a data set name conforms to the conventions, you can refer to the data set by its fully qualified name or by an abbreviated version of the name. The following paragraphs:

1. Describe data set names in general.
2. Define the names that conform to the naming conventions for TSO.

3. Tell you how to enter a complete data set name, and how to enter the abbreviated version of a name that conforms to the TSO data set naming conventions.

Data Set Names in General

A data set name consists of one or more fields. Each field consists of one through eight alphameric characters and must begin with an alphabetic (or national) character.

CAUTION: The National Characters are $, @, and # are accepted as the first character in a data set name. The characters hyphen (-) and ampersand-zero (12-0 punch) are not accepted in a data set name.

A simple data set name with only one field may be:

PARTS

A data set name that consists of more than one field is a "qualified" data set name. The fields in a qualified data set name are separated by periods. A qualified data set name may be:

PARTS.OBJ
or
PARTS.DATA

Partitioned Data Sets: A partitioned data set is simply a data set with the data divided into one or more independent groups called members. Each member is identified by a member name and can be referred to separately. The member name is enclosed within parentheses and appended to the end of the data set name:

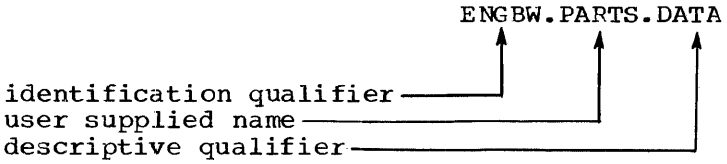
PARTS.DATA(PART14)
member name

TSO Data Set Names

A data set name must be qualified in order to conform to the TSO data set naming conventions. The qualified name must consist of at least the two required fields of the following three:
1. Your user identification (required).
2. A user-supplied name (optional for a partitioned data set).
3. A descriptive qualifier (required).

Normally all three names are used.

The total length of the data set name must not exceed 44 characters, including periods. A typical TSO data set name is:



The TSO data set naming conventions also apply to partitioned data sets. A typical TSO name for a member of a partitioned data set is:

ENGBW.PARTS.DATA(PART14)

Identification Qualifier: The identification qualifier is always the leftmost qualifier of the full data set name. For TSO, this qualifier is the user identification assigned to you by your installation.

User-supplied Name: You choose a name for the data sets that you want to identify. It can be a simple name or several simple names separated by periods.

Descriptive Qualifier: The descriptive qualifier is always the rightmost qualifier of the full data set name. To conform to the data set naming conventions, this qualifier must be one of the qualifiers listed in Figure 3.

Descriptive Qualifier	Data Set Contents
ASM	Assembler (F) input
BASIC	ITF:BASIC statements
CLIST	TSO commands
CNTL	JCL and SYSIN for SUBMIT command
COBOL	American National Standard COBOL statements
DATA	Uppercase text
FORT	FORTRAN (Code and Go, E, G, G1, H) statements
IPLI	ITF:PL/I statements
LINKLIST	Output listing from linkage editor
LIST	Listings
LOAD	Load module
LOADLIST	Output listing from loader
OBJ	Object module
OUTLIST	Output listing from OUTPUT command
PLI	PL/I (F) statements and PL/I Checkout and Optimizing compiler statements.
STEX	STATIC external data from ITF:PLI
TESTLIST	Output listing from TEST command
TEXT	Uppercase and lowercase text

Figure 3. Descriptive Qualifiers

How to Enter Data Set Names

The data set naming conventions simplify the use of data set names. If the data set name conforms to the conventions, you need specify only the user-supplied name field (in most cases) when you refer to the data set. The system will add the necessary qualifiers to the beginning and to the end of the name that you specify. In some cases, however, the system will prompt you for a descriptive qualifier. Until you learn to anticipate these exceptions to the naming conventions, you may wish to specify both the user-supplied name and the descriptive qualifier when referring to a data set. When you are using the LINK command for example, the system will add both the user identification and the descriptive qualifier, allowing you to specify only the user-supplied name. For instance, you may refer to the data set named USERID.PARTS.OBJ by specifying only PARTS (when you are using LINK) or by specifying PARTS.OBJ (when you are using other commands). You may refer to a member of a partitioned data set USERID.PARTS.OBJ(PART14) by specifying PARTS(PART14) when you are using LINK or by specifying PARTS.OBJ(PART14) when you are using other commands.

When you specify an entire fully qualified data set name, as you must do if the name does not conform to the TSO data set naming conventions, you must enclose the entire name within apostrophes; as follows:

'JOED58.PROG.LIST'	where JOED58 is not your user identification
or	
'JOED58.PROG.FIRST'	where FIRST is not a valid descriptive qualifier.

The system will not append qualifiers to any name enclosed in parentheses.

Defaults for Data Set Names: When you specify only the user-supplied name, the system adds your user identification and, whenever possible, a descriptive qualifier. The system attempts to derive the descriptive qualifier from available information. For instance, if you specified ASM as an operand for the EDIT command, the system will assign ASM as the descriptive qualifier. If the information is insufficient, the system will issue a message at your terminal requesting the required information. If you specify the name of a partitioned data set and do not include a required member name, the system will use TEMPNAME as the default member name. (If you are creating a new member, the member name will become TEMPNAME; if you are modifying an existing partitioned data set, the system will search for a member named TEMPNAME.) The following example illustrates the default names supplied by the system.

If you specify:	The input data set name is:	The output data set name will be:
EDIT PARTS ASM	UID.PARTS.ASM	UID.PARTS.ASM
LINK PARTS or		
LINK(PARTS)	UID.PARTS.OBJ	UID.PARTS.LOAD(TEMPNAME)
CALL PARTS	UID.PARTS.LOAD(TEMPNAME)	---
EDIT PARTS(JAN) ASM	UID.PARTS.ASM(JAN)	UID.PARTS.ASM(JAN)
LINK PARTS(JAN) or		
LINK (PARTS(JAN))	UID.PARTS.OBJ(JAN)	UID.PARTS.LOAD(JAN)
CALL PARTS(JAN)	UID.PARTS.LOAD(JAN)	---
EDIT (PARTS) ASM	UID.ASM(PARTS)	UID.ASM(PARTS)
LINK ((PARTS))	UID.OBJ(PARTS)	UID.LOAD(PARTS)
CALL (PARTS)	UID.LOAD(PARTS)	---

Note: In these examples, UID stands for your user identification. TEMPNAME is the membername supplied by the system.

Note: Member names must be enclosed in parentheses to distinguish them from data set names. Figure 4 presents a list of command names and the default descriptive qualifiers associated with each command.

Command	DESCRIPTIVE QUALIFIERS Input	Output	Listing
ASM	ASM	OBJ	LIST
CALC	STEX	STEX	---
CALL	LOAD	---	---
COBOL	COBOL	OBJ	LIST
CONVERT	IPLI	PLI	---
	FORT	FORT	---
EXEC	CLIST	---	---
FORMAT	TEXT	---	LIST
FORT	FORT	OBJ	LIST
LINK	OBJ	LOAD	LINKLIST
	LOAD	---	---
LOADGO	OBJ	---	LOADLIST
	LOAD	---	---
OUTPUT	---	---	OUTLIST
RUN	ASM	---	---
	FORT	---	---
	BASIC	---	---
	COBOL	---	---
	IPLI	---	---
SUBMIT	CNTL	---	---
TEST	OBJ	---	TESTLIST
	LOAD	---	---

Figure 4. Descriptive Qualifiers Supplied by Default

Specifying Data Set Passwords

When referencing password protected data sets, you may specify the password as part of the data set name (you will be prompted for it otherwise). The password is separated from the data set name by a slash (/) and optionally, by one or more standard delimeters (tab, blank, or comma). See the discussion on "Password Data set" that appears under the PROTECT command.

System-Provided Aids

Several aids are available for your use at the terminal:

- The attention interruption allows you to interrupt processing so that you can enter a command.
- The HELP command provides you with information about the commands.
- The conversational messages guide you in your work at the terminal.

The Attention Interruption

The attention interruption allows you to interrupt processing at any time so that you can enter a command or subcommand. For instance, if you are executing a program and the program gets in a loop, you can use the attention interruption to halt execution. As another example, when you are having the data listed at your terminal and the data that you need has been listed, you may use the attention interruption to stop the listing operation instead of waiting until the entire data set has been listed.

If, after causing an attention interruption, you want to continue with the operation that you interrupted, you can do so by pressing the return key before typing anything else; however, input data that was being typed or output data that was being printed at the time of the attention interruption may be lost. You can also request an attention interruption while at the command level, enter the TIME command, and then resume with the interrupted operation by pressing the return key.

If your terminal has an interruption facility, you can request an attention interruption by pressing the appropriate key (the ATTN key on IBM 2741 Communications Terminals). Whether or not your terminal has a key for attention interruptions, you can use the TERMINAL command to specify particular operating conditions that the system is to interpret as a request for an attention interruption. More specifically, you can specify a sequence of characters that the system is to interpret as a request for an attention interruption. In addition, you can request the system to pause after a certain number of seconds of processing time has elapsed or after a certain number of lines of output has been displayed at your terminal. When the system pauses, you can enter the sequence of characters that you define as a request for an attention interruption.

Note: If you are using the attention key as a line-delete indicator, pressing the attention key (after entering characters in a line, and before pressing the carriage return,) will cause the line you entered to be ignored by the system. Another depression of the attention key is required to cause an interruption.

The HELP Command

The HELP command provides you with information about the use, function, syntax, and operands of commands and subcommands. When you enter HELP, the system displays at your terminal a list of commands and a brief description of the function of each. By specifying a command name as an operand for the HELP command, you can get a list of operands and a description of the function and syntax of the command.

HELP is also a subcommand for all of the commands that have subcommands. By specifying a subcommand name as an operand for the HELP

subcommand, you can get a list of operands and a description of the function and syntax of the subcommand.

Message Types

You receive three types of messages at your terminal:

Mode messages.
Prompting messages.
Informational messages.

A mode message tells you the system is ready to accept new input -- a command, a subcommand, or data. When the system is waiting for you to enter a command, the mode message displayed at your terminal is:

READY

Other mode messages may be displayed, when appropriate, to tell you that the system is waiting for you to enter a subcommand or data. In these cases, the mode message is the name of the current command or subcommand:

ACCOUNT
EDIT
INPUT
OPERATOR
OUTPUT
TEST
etc.

These mode messages are displayed when the mode changes.

A prompting message tells you that required information is missing and that you must take an explicitly described action in response. For instance, prompting messages prompt you to supply missing operands and to correct operands that you specified incorrectly. A typical prompting message is:

ENTER DATA SET NAME-

The system expects an immediate response to messages that end with a hyphen. Use the PROMPT or NOPROMPT operand of the PROFILE command to specify whether or not you want to receive prompting messages. You can stop a prompting sequence by requesting an attention interruption.

An informational message tells you about the status of the system and your terminal session. For instance, an informational message may tell you when program execution has terminated, or how much time you have used. Informational messages do not require a response.

In some cases, an informational message may serve as a mode message; for instance, an informational message that tells of the completion of a subcommand's operation also implies that you can enter another subcommand.

Levels of Messages: Prompting messages and informational messages may have additional messages associated with them. The additional messages explain the initial message more fully.

Prompting messages may have any number of additional messages; informational messages may have only one additional message. When an

additional informational message is available, the message at your terminal will end with a plus sign (+); prompting messages do not end with a plus sign, even though an additional message may be available.

The Question Mark: To receive an additional message, you must enter a question mark (?) and a carrier return. When you enter a question mark, it must be placed in the first position on the line. You can continue entering question marks until no other message is available. When no other message exists, the system will display:

NO INFORMATION AVAILABLE

For example, a listing at your terminal may look like:

```
INVALID LINE NUMBER ENCOUNTERED+
?
USE EDIT WITH NONUM OPERAND
?
NO INFORMATION AVAILABLE
```

This section contains descriptions of the TSO commands. The commands are presented in alphabetical order. Subcommands are presented in alphabetical order following the command to which they apply.

COMMAND (Abbreviation) SUBCOMMAND (abbreviation)	COMMAND(Abbreviation) SUBCOMMAND(Abbreviation)
ACCOUNT	LOGOFF
ADD (A)	LOGON
CHANGE (C)	*MERGE
DELETE (D)	OPERATOR (OPER)
END	CANCEL (C)
HELP (H)	DISPLAY (D)
LIST (L)	END
LISTIDS (LISTI)	HELP (H)
ALLOCATE (ALLOC)	MODIFY (F)
*ASM	MONITOR (MN)
ATTRIB (ATTR)	SEND
*CALC	STOPMN (PM)
CALL	OUTPUT (OUT)
CANCEL	CONTINUE (CONT)
*COBOL (COB)	END
*CONVERT (CON)	HELP (H)
*COPY	SAVE (S)
DELETE (D)	PROFILE (PROF)
EDIT (E)	PROTECT (PROT)
BOTTOM (B)	RENAME (REN)
CHANGE (C)	RUN (R)
DELETE (D)	SEND (SE)
DOWN	STATUS (ST)
EDIT (E)	SUBMIT (SUB)
END	TERMINAL (TERM)
FIND (F)	TEST
*FORMAT (FORM)	Assignment of Values ( )
HELP (H)	AT
INPUT (I)	CALL
INSERT (IN)	COPY(C)
Insert/Replace/Delete ( )	DELETE (D)
LIST (L)	DROP
*MERGE (M)	END
PROFILE (PROF)	EQUATE (EQ)
RENUM (REN)	FREEMAIN (FREE)
RUN (R)	GETMAIN (GET)
SAVE (S)	GO
SCAN (SC)	HELP (H)
TABSET (TAB)	LIST (L)
TOP	LISTDCB
UP	LISTDEB
VERIFY (V)	LISTMAP
EXEC (EX)	LISTPSW
*FORMAT (FORM)	LISTTCB
*FORT	LOAD
FREE	OFF
HELP (H)	QUALIFY (Q)
LINK	RUN (R)
*LIST (L)	WHERE (W)
LISTALC (LISTA)	TIME
LISTBC (LISTB)	
LISTCAT (LISTC)	
LISTDS (LISTD)	
LOADGO (LOAD)	

*Optional Program Product commands available for a license fee (Appendix C).
**For use in command procedures.

ACCOUNT Command

Use the ACCOUNT command and subcommands to create and to update the entries in the User Attribute Data Set (UADS) and, simultaneously, the Broadcast Data Set. (You can use this command only if your installation has given you the authority to do so.) Basically, the UADS is a list of terminal users who are authorized to use TSO. The UADS contains information about each of the users. The information in the UADS is used to regulate access to the system.

COMMAND	OPERANDS
ACCOUNT	

Subcommands of ACCOUNT

You cannot accomplish any work with the ACCOUNT command until you use a subcommand to define the operation that you want to perform. The subcommands and the operations that they define are:

ADD	Add new entries to the UADS; add new data to existing entries.
CHANGE	Change data in specific fields of UADS entries.
DELETE	Delete entries or parts of entries from the UADS.
END	Terminate the ACCOUNT command.
HELP	Obtain help from the system.
LIST	Display the contents of an entry in the UADS.
LISTIDS	Display the user identifications for all entries.

The subcommands cannot be used until you have entered the ACCOUNT command. Each subcommand is discussed separately following the format of the ACCOUNT command.

There is an entry in the UADS for each terminal user. Each entry consists of the following information:

1. A user identification.
2. One or more passwords, or a single null field, associated with the user identification.
3. One or more account numbers, or a single null field, associated with each password.
4. One or more procedure names associated with each account number. Each procedure name identifies a procedure that is invoked when the user begins a terminal session by entering the LOGON command.
5. The region size requirements for each procedure.
6. The name of the group of devices that the user will be permitted to use for a procedure. Data sets allocated via the catalog are an exception. See the ALLOCATE command.

7. The authority to use or a restriction against using the ACCOUNT command.

8. The authority to use or a restriction against using the OPERATOR command.

9. The authority to use or restriction against using the SUBMIT, STATUS, CANCEL, and OUTPUT commands.

10. Maximum region size allowed.

The organization of the information contained in the UADS is shown in Figure 5. Figure 6 shows the simplest structure that an entry in the UADS can have, and Figure 7 shows a more complex structure.

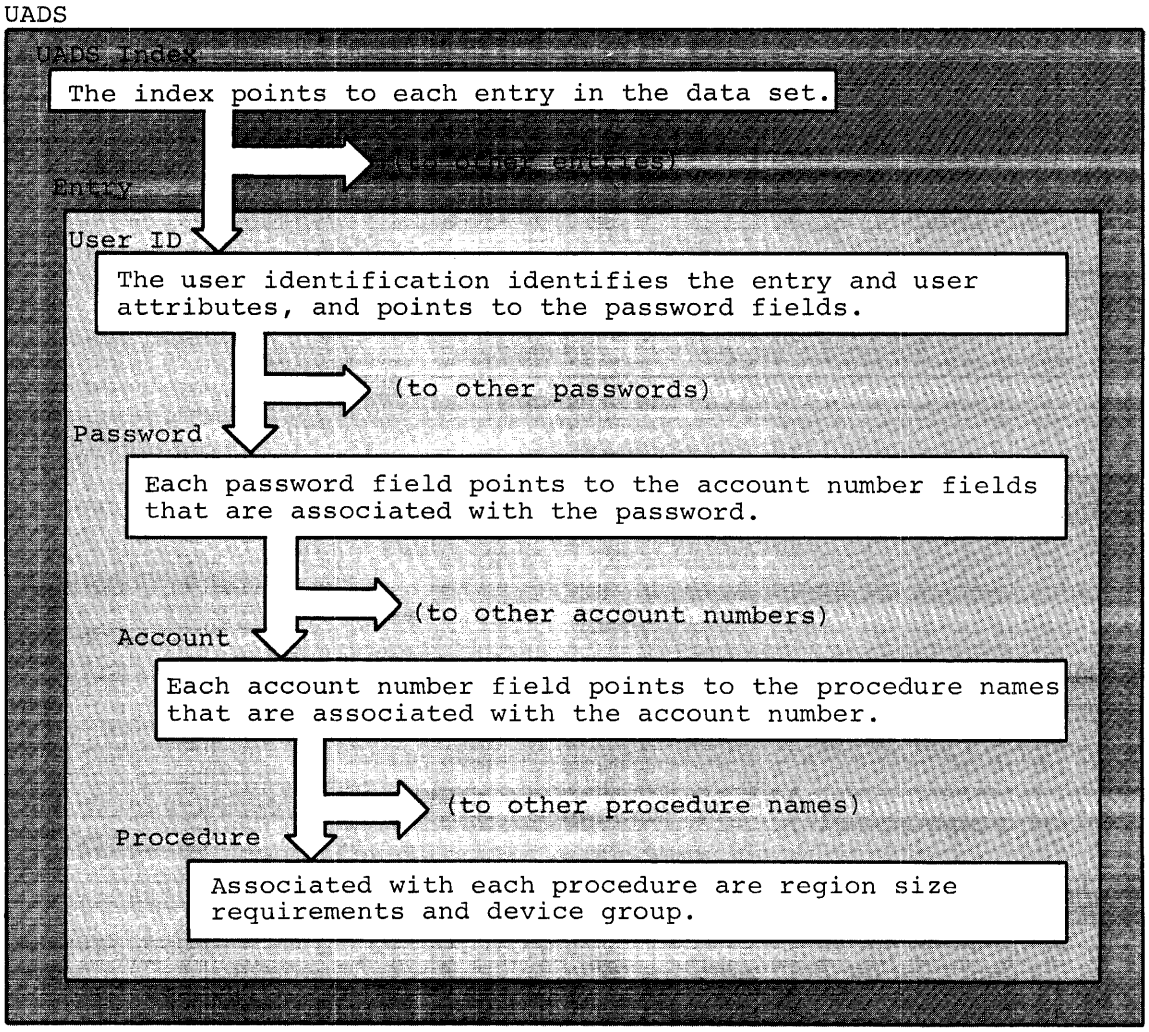


Figure 5. Organization of the UADS Data Set

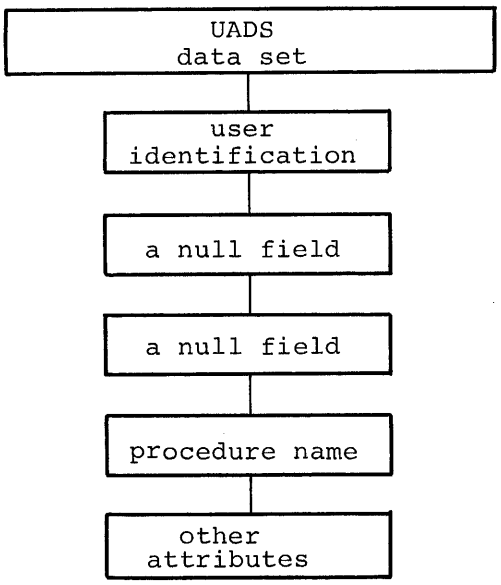


Figure 6. The Simplest Structure That an Entry in the UADS Can Have

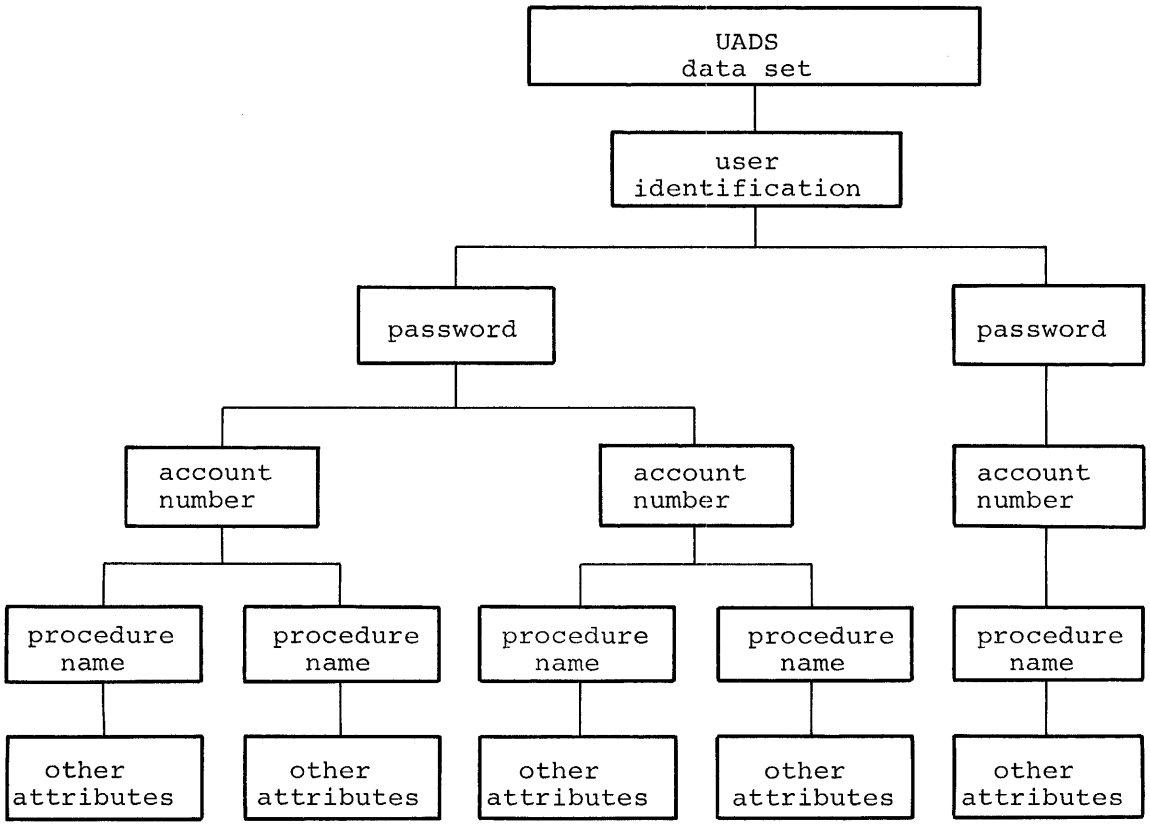


Figure 7. A Complex Structure For an Entry in the UADS

ADD Subcommand of ACCOUNT

Use the ADD subcommand to create new userids for prospective users of TSO. As you create a new userid, a corresponding entry is created in the User Attribute Data Set (UADS) for that user (see Figures 5, 6, and 7). For each new userid that you create, the system builds a "typical" user profile in the User Profile Table (UPT) for that user. This "typical" user profile is discussed under the PROFILE command in this publication.

You can also use ADD to add additional data to an existing entry in the UADS. Do not use ADD to change any existing data in a UADS entry; use the CHANGE subcommand instead.

When adding a new entry to the UADS, you can also select the following options for the new user:

- The region size that he can request at LOGON
- The authority to use the ACCOUNT command.
- The authority to use the OPERATOR command.
- The authority to use the SUBMIT, STATUS, CANCEL, and OUTPUT commands.

SUBCOMMAND	OPERANDS
{ADD / A}	({user-identity / *} [password [account [procedure]]])
	[DATA([[passwords]accounts]procedures)]
	[SIZE(integer)]
	[UNIT(name)]
	[MAXSIZE(integer) / NOLIM]
	[ACCT / NOACCT] [OPER / NOOPER] [JCL / NOJCL]

user identity
specifies a user identification that identifies the UADS entry. The user identification is composed of 1-7 alphameric characters that begin with an alphabetic or national character. The entry that this field identifies may be:

- An existing entry to which new data is to be added.
- A new entry that is to be added to the UADS.

*
specifies that all fields corresponding to the position of the asterisk are to be considered valid for the operation of the subcommand. When you are creating a new entry, the asterisk indicates a null field.

password
specifies a word that the user must enter before he can use the system. The word must be composed of 1-8 alphameric characters. The password helps indicate the structure in the UADS to which data is being added, or, when you are adding an entire new entry, the password is part of the data being added.

account
specifies an account number used for administrative purposes. The account number helps indicate the structure in the UADS to which data is being added, or, when you are adding an entire new entry, the account number is part of the data being added.

For TSO, an account number must not exceed 40 characters, and must not contain a blank, tab, quotation mark, apostrophe, comma, semicolon, or line-control character. A right parenthesis is permissible only when a left parenthesis precedes it somewhere in the account number.

procedure
specifies the name of a procedure that is invoked when the user enters the LOGON command. The procedure name is composed of 1-8 alphameric characters that begin with an alphabetic character. You should not specify this field for the first positional operand unless you are adding an entire new entry to the UADS.

DATA(passwords and/or accounts and/or procedures)
specifies that data is to be added to an existing entry. The data to be added is enclosed within parentheses following the DATA keyword. The system adds the data specified with this keyword to the structure identified by the positional operand. The data is added to the next lower level in the existing structure. The complexity of the positional operand "user identity" determines how many levels of structure exist.

passwords
specifies a password or a list of passwords to be added to the existing entry at the location indicated by the positional operand. When you specify a list of passwords, the list must be enclosed within a separate set of parentheses embedded within the set of parentheses required for the DATA keyword. Each password must be composed of 1-8 alphameric characters.

accounts
specifies an account number or a list of account numbers to be added to the existing entry. When you specify a list of account numbers, the list must be enclosed within a separate set of parentheses embedded within the set of parentheses required for the DATA keyword. An account number must not exceed 40 characters and must not contain a blank, tab, quotation mark, semicolon, or line control character; a right parenthesis is permissible only when a left parenthesis balances it somewhere in the account number. No more than 255 identical account numbers may exist under one user entry.

procedures
specifies a procedure name or a list of procedure names to be added to the existing entry. Each procedure name is composed of 1-8 alphameric characters that begin with an alphabetic character. When you specify a list of procedure names, in addition to one or more other fields, the list must be enclosed within a separate set

of parentheses embedded within the set of parentheses required for the DATA keyword. You should specify the region size requirements for each procedure by using the SIZE keyword. No more than 255 identical procedure names may exist under one user entry.

SIZE(integer)
specifies the minimum region size, in 1024 byte units, that the user will have assigned to him if he does not specify a size himself. The integer specified must not exceed 65,534. If you omit the SIZE keyword or if you specify SIZE(0), the default value is the minimum region size available.

UNIT(name)
specifies the name of the group of devices that the user (identified by the positional operand) will use. Data sets allocated via the catalog are an exception. See the ALLOCATE command. You can specify a UNIT attribute for each unique combination of password, account, and procedure in the entry.

MAXSIZE(integer)
specifies the maximum region size, in 1024 byte units, that the user (identified by the first operand) can request at LOGON. The integer must not exceed 65,534. If you omit the MAXSIZE keyword or if you specify MAXSIZE(0), the default of NOLIM is assumed. Use this operand only when you add a complete entry to the UADS.

NOLIM
If NOLIM is specified, no maximum region size limit is enforced. This is the default when neither MAXSIZE nor NOLIM is specified. Use this operand only when you add a complete entry to the UADS.

ACCT
specifies that the user (identified by the first operand) can use the ACCOUNT command, thereby controlling access to the time sharing system. Use this operand only when you add a complete entry to the UADS.

NOACCT
specifies that the user (identified by the first operand) cannot use the ACCOUNT command. This is the default when neither ACCT nor NOACCT is specified. Use this operand only when you add a complete entry to the UADS.

OPER
specifies that the user (identified by the first operand) can use the OPERATOR command. Use this operand only when you add a complete entry to the UADS.

NOOPER
specifies that the user (identified by the first operand) cannot use the OPERATOR command. This is the default when neither OPER nor NOOPER is specified. Use this operand only when you add a complete entry to the UADS.

JCL
specifies that the user (identified by the first operand) can use the SUBMIT, STATUS, CANCEL, and OUTPUT commands. Use this operand only when you add a complete entry to the UADS.

NOJCL
specifies that the user (identified by the first operand) cannot use the SUBMIT, STATUS, CANCEL, and OUTPUT commands. This is the default when neither JCL nor NOJCL is specified. Use this operand only when you add a complete entry to the UADS.

Example 1

Operation: Add a new entry to the UADS.

Known: The user identification..... KALTPT
The password..... XAYBZC
The account number..... 32058
The procedure name..... MYLOG
The user cannot use the ACCOUNT command.
The user cannot use the OPERATOR command.
The user can use the SUBMIT command.
The user's maximum allowable region size..... 153,600 bytes
The region size requirements for the procedure... 81,920 bytes
The name of the group of devices allowed..... SYSDA

```
ADD (KALTPT XAYBZC 32058 MYLOG) NOACCT NOOPER JCL -
MAXSIZE(150) SIZE(80) UNIT(SYSDA)
```

Example 2

Operation: Add a new password, account number, and procedure name to an existing entry in the UADS. Also include the region size requirements for the procedure.

Known: The user identification for the entry..... SLAT2
The new password..... MZ3TII
The new account number..... 7116166
The new procedure name..... AMABALA
The region size requirements for the procedure... 92,160 bytes

```
ADD (SLAT2) DATA(MZ3TII 7116166 AMABALA) SIZE(90)
```

Example 3

Operation: Continuing example 2, add a new account number, 288104, to an existing entry in the UADS.

Known: The user identification for the entry..... SLAT2
The password for the entry..... MZ3TII
The new account number..... 288104
The new procedure name..... MYLOG
The region size requirements for the procedure... 116,736 bytes
The device group to be used..... SYSDA

```
ADD (SLAT2 MZ3TII) DATA(288104 MYLOG) SIZE(114) UNIT(SYSDA)
```

Example 4

Operation: Add a new procedure name, and the region size requirements for it, to all entries in the UADS.

Known: The new procedure name..... MYLOG
The region size requirements for it..... 74,752 bytes

```
ADD (* * *) DATA(MYLOG) SIZE(73)
```

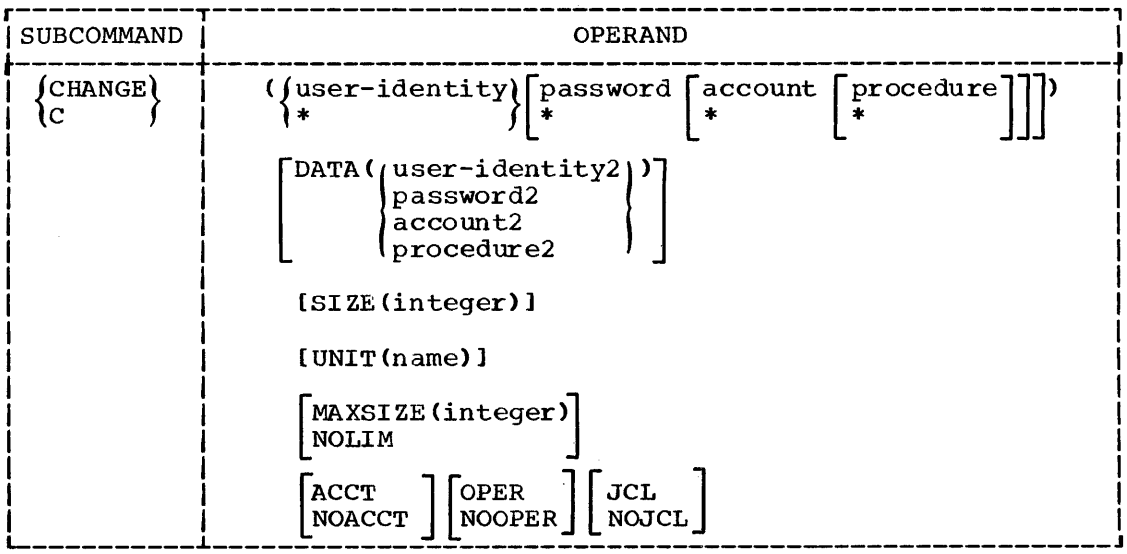
Example 5

Operation: Add a new account number and new procedure name to all structures under an existing entry in the UADS.

Known: The user identification for the entry..... WMROEL
The new account number..... 5707471
The new procedure name..... LOGPROC
The region size requirements..... 102,400 bytes

```
ADD (WMROEL *) DATA(5707471 LOGPROC) SIZE(100)
```

Use the CHANGE subcommand to change existing fields of data within entries in the UADS.



user-identity
specifies the existing user identification that identifies the UADS entry that is to be changed.

*
specifies that all fields corresponding to the position of the asterisk are to be considered valid for the operation of the subcommand.

password
specifies an existing password that a user must enter before he can use the system. The password helps locate the data being changed, and, when you are changing a password, identifies the password being changed. A password must consist of from 1 to 8 alphameric characters.

account
specifies an existing account number. The account number helps locate the data being changed, and, when you are changing an account number, identifies the account number being changed.

procedure
specifies an existing name of a procedure. The procedure name, when specified, is the data being changed.

DATA(user identity2 and/or password2 and/or account2 and/or procedure2)
specifies the replacement data. The data enclosed within parentheses following the DATA keyword is used by the system to replace the data identified by the last field of the first operand.

user identity2
specifies a user identification to replace the existing user identity. The user identification is composed of 1-7 alphameric characters that begin with an alphabetic or national character.

password2
specifies a password to replace the existing password. The password must be composed of 1-8 alphameric characters.

account2
specifies an account number to replace the existing account number. The account number is composed of 1-40 characters and must not contain a blank, tab, quotation mark, semicolon, apostrophe, comma, or line control character. A right parenthesis is permissible only when a left parenthesis balances it somewhere in the account number.

procedure2
specifies a procedure name to replace the existing procedure name. The procedure name is composed of 1-8 alphameric characters and must begin with an alphabetic character.

SIZE(integer)
specifies the region size, in 1024 byte units, that is specified on the JCL EXEC statement of the procedure whose name is being added to the UADS. The integer must not exceed 65,534. If you specify SIZE(0), the minimum region size is assumed.

UNIT(name)
specifies the name of the group of devices that the user (identified by the first operand) will use. Data sets allocated via the catalog are an exception. See the ALLOCATE command.

MAXSIZE(integer)
specifies the maximum region size, in 1024 byte units, that the user may request at LOGON. The integer must not exceed 65,534. If you specify MAXSIZE(0), the default of NOLIM is assumed.

NOLIM
specifies that the user is not restricted to a maximum region size.

ACCT
specifies that the user can use the ACCOUNT command thereby controlling access to the time sharing system.

NOACCT
specifies that the user cannot use the ACCOUNT command.

OPER
specifies that the user can use the OPERATOR command.

NOOPER
specifies that the user cannot use the OPERATOR command.

JCL
specifies that the user can use the SUBMIT, STATUS, CANCEL, and OUTPUT commands.

NOJCL
specifies that the user cannot use the SUBMIT, STATUS, CANCEL, and OUTPUT commands.

Example 1

Operation: Change an account number for an entry in the UADS and authorize the user to issue the ACCOUNT and OPERATOR commands.

Known: The user identification for the entry..... TOC23
The password..... AOX3P
The old account number..... 2E29705
The new account number..... 2E26705

```
CHANGE (TOC23 AOX3P 2E29705) DATA(2E26705) ACCT OPER
```

Example 2

Operation: Authorize all users to issue the SUBMIT command.

```
CHANGE (*) JCL
```

The asterisk in the first positional operand position specifies that all user identities are considered valid for the operation of this subcommand.

Example 3

Operation: Change the user identification for an entry in the UADS.

Known: The existing user identification..... SWECORP
The new user identification..... SWECP01

```
CHANGE (SWECORP) DATA(SWECP01)
```

Example 4

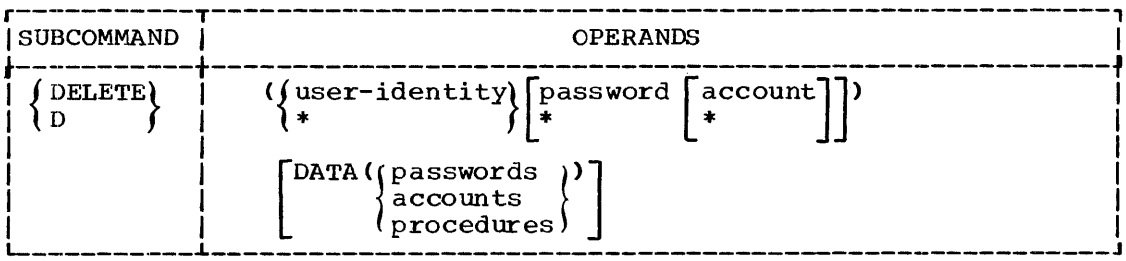
Operation: Change the name of a procedure for an entry that consists of a user identification, a procedure name, and attributes (no password or account number).

Known: The user identification..... WSNCD
The old procedure name..... TTURM
The new procedure name..... TML

```
CHANGE (WSNCD * * TTURM) DATA(TML)
```

DELETE Subcommand of ACCOUNT

Use the DELETE subcommand to delete data from the User Attribute Data Set (UADS). Each terminal user has an entry in the UADS. Each entry contains several items of data. The data that you want to delete may be a part of an existing entry, or it may be an entire existing entry.



user-identity
specifies a user identification which identifies the UADS entry that is to be deleted. The user identification is composed of 1-7 alphameric characters that begin with an alphabetic or national character.

*
specifies that all fields corresponding to the position of the asterisk are to be considered valid for the operation of the subcommand.

password
specifies a word that a user must enter before he can use the system. The word must be composed of 1-8 alphameric characters. The password helps indicate the particular existing structure from which data is being deleted, or, when you are deleting a password, the password is the data being deleted.

account
specifies an account number used for administrative purposes. The account number helps indicate the structure from which data is being deleted, or, when you are deleting an account number, the account number is the data being deleted.

For TSO, an account number must not exceed 40 characters, and must not contain a blank, tab, quotation mark, semicolon, apostrophe, comma, or line control character. A right parenthesis is permissible only when a left parenthesis precedes it somewhere in the account number.

DATA(passwords or accounts or procedures)
specifies the data that is to be deleted from an existing entry. The data to be deleted is enclosed within parentheses following the DATA keyword.

passwords
specifies a password or a list of passwords to be deleted from the existing entry at the location indicated by the first positional operand. Each password must be composed of 1-8 alphameric characters.

accounts
specifies an account number or a list of account numbers to be deleted from the existing entry. An account number must not exceed 40 characters, and must not contain a blank, tab, quotation mark, apostrophe, comma, semicolon, or line control character. A right parenthesis is permissible only when a left parenthesis balances it somewhere in the account number.

procedures
specifies a procedure name or a list of procedure names to be deleted from the existing entry. Each procedure name is composed of 1-8 alphameric characters and must begin with an alphabetic character.

The Contents of an Entry in the UADS: Each entry in the UADS consists of the following information:

(These five items that follow correspond to the fields of the first positional operand and the DATA keyword for this subcommand. These items are the only items that you can delete separately. To delete items 6-9, you must delete the entire entry.)

1. A user identification.

2. One or more passwords, or a single null field, associated with the user identification.

3. One or more account numbers, or a single null field, associated with each password.

4. One or more procedure names associated with each account number. Each procedure name identifies a procedure that is invoked when the user begins a terminal session by entering the LOGON command.

5. The region size requirements for each procedure.

(These last four items can be deleted only when the entire entry is deleted.)

6. The name of the group of devices that the user will be permitted to use. Data sets allocated via the catalog are an exception. See the ALLOCATE command.

7. The authority to use, or a restriction against using, the ACCOUNT command.

8. The authority to use, or a restriction against using, the OPERATOR command.

9. The authority to use, or a restriction against using, the SUBMIT, STATUS, CANCEL, and OUTPUT commands.

Deleting an Entire Entry: To delete an entire entry from the UADS, you only need to know the user identification for the entry. You must specify the user identification as the first and only field of the first positional operand.

Deleting Data from an Existing Entry: To use the DELETE subcommand to delete data from an existing entry, you must identify:

a. The location within the entry.
b. The data that you want to delete.

Example 1

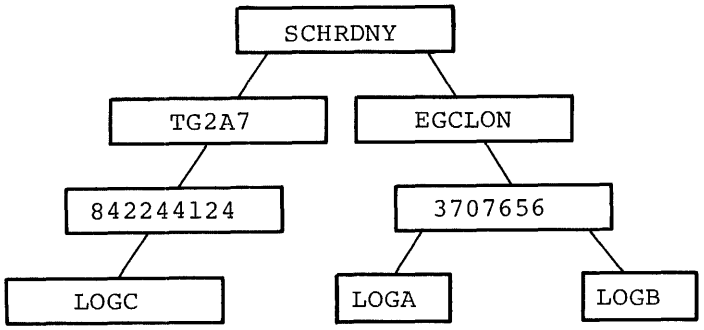
Operation: Delete an entire entry from the UADS.

Known: The user identification for the entry..... VASHTAR

```
DELETE (VASHTAR)
```

Example 2

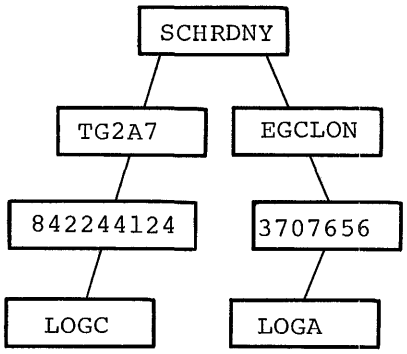
Operation: Delete a procedure name from an entry in the UADS having the following index structure.



Known: The user identification..... SCHRDNY
The password..... EGCLON
The account number..... 3707656
The procedure name to be deleted..... LOGB

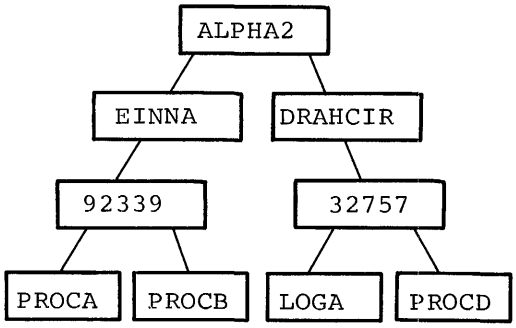
```
DELETE (SCHRDNY EGCLON 3707656) DATA(LOGB)
```

The resultant index structure is:



Example 3

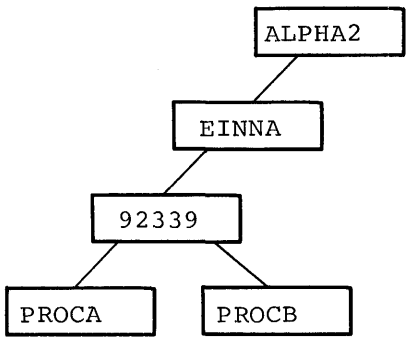
Operation: Delete an account number from an entry in the UADS having the following index structure.



Known: The user identification..... ALPHA2
The password..... DRAHCIR
The account number to be deleted..... 32757

```
DELETE (ALPHA2 DRAHCIR) DATA(32757)
```

The resultant index structure is:



END Subcommand of ACCOUNT

Use the END subcommand to terminate operation of the ACCOUNT command. After entering the END subcommand, you may enter new commands.

SUBCOMMAND	OPERANDS
END	

Use the HELP subcommand to find out how to use ACCOUNT and the ACCOUNT subcommands. When you enter the HELP subcommand, the system responds by printing out explanatory information at your terminal. You may request:

- A list of available subcommands.
- An explanation of the function, syntax, and operands of a specific subcommand.

The HELP subcommand actually causes the system to execute a function of the HELP command; therefore, you may consult the discussion of the HELP command if you desire more detailed information.

SUBCOMMAND	OPERANDS
{HELP / H}	[subcommand-name [FUNCTION / SYNTAX / OPERANDS(list-of-operands) / ALL]]

subcommand-name
specifies the subcommand that you want to have clarified. If you omit this operand, the system will display a list of ACCOUNT subcommands.

FUNCTION
specifies that you want a description of the referenced subcommand's function.

SYNTAX
specifies that you want a definition of the proper syntax for the referenced subcommand.

OPERANDS(list-of-operands)
specifies that you want an explanation of the operand applicable to the referenced subcommand.

The list of operands specifies the particular keywords that you want to have explained. If you do not specify any keywords, all of the applicable keywords will be included. You must use one or more blanks or a comma as a delimiter between the keywords in the list.

ALL
specifies that you want a description of the function, the syntax, and the operands of the subcommand that you specified. This is the default if no operand is specified.

Example 1

Operation: Have a list of available subcommands displayed at your terminal.

```
HELP
```

Example 2

Operation: Obtain all available information about a particular subcommand.

Known: The subcommand name..... ADD

```
H ADD
```

Example 3

Operation: Have a list of the operands for a particular subcommand displayed at your terminal.

Known: The subcommand name..... LIST

```
h list operands
```

Use the LIST subcommand to display entries in the User Attribute Data Set (UADS) or to display fields of data from within particular entries.

SUBCOMMAND	OPERANDS
{LIST / L}	({user-identity / *}[password [account [procedure]]])

user-identity
specifies a user identification that identifies the UADS entry. The user identification is composed of 1-7 alphameric characters that begin with an alphabetic or national character.

*
specifies that all fields corresponding to the position of the asterisk are to be considered valid for the operation of the subcommand.

password
specifies a word that a user must enter before he can use the system. The word must be composed of 1-8 alphameric characters. The password helps indicate the structure to be displayed.

account
specifies an account number used for administrative purposes. The account number helps indicate the structure to be displayed. For TSO, an account number must not exceed 40 characters, and must not contain a blank, tab, quotation mark, apostrophe, comma, semicolon, or line control character. A right parenthesis is permissable only when a left parenthesis precedes it somewhere in the account number.

procedure
specifies the name of a procedure that is invoked when the user enters the LOGON command. The procedure name helps indicate the particular structure to be displayed. The procedure name is composed of 1-8 alphameric characters and must begin with an alphabetic character.

Example 1

Operation: List the contents of the UADS.

```
LIST (*)
```

Example 2

Operation: List all of a particular entry in the UADS.

Known: The user identification..... JOTSOP

```
LIST (JOTSOP)
```

Example 3

Operation: List all of the account numbers under a specific password for a particular entry.

Known: The user identifcation..... EVOTS
The password..... ROOLF

```
LIST (EVOTS ROOLF *)
```

Use the LISTIDS subcommand to have a list of the user identifications in the User Attribute Data Set (UADS) displayed at your terminal.

SUBCOMMAND	OPERANDS
{LISTIDS {LISTI	

Example 1

Operation: List all user identifications in the UADS.

```
LISTIDS
```

ALLOCATE Command

Use the ALLOCATE command to allocate, dynamically, the data sets required by a program that you intend to execute.

COMMAND	OPERANDS
ALLOCATE ALLOC	{DATASET({* data-set-name}) [FILE(name)] FILE(name) [DATASET({* data-set-name})]} [OLD SHR MOD NEW SYSOUT] [VOLUME(serial)] [SPACE(quantity [increment]) BLOCK(block-length)] [DIR(integer)] [USING(attribute-list-name)]

DATASET(data-set-name, data-set-name/password, or *)
specifies the name of the data set that is to be allocated. The data set name must include the descriptive (rightmost) qualifier and may contain a member name in parentheses. (See the data set naming conventions.)

If you specify a password, you will not be prompted for it when you open the data set. Any other user, however, must supply the password in order to refer to the data set.

You may substitute an asterisk (*) for the data set name to indicate that you want to have your terminal allocated for input and output. If you use an asterisk (*), only the FILE operand is recognized by the system. All other operands are ignored.

In general, you may specify either or both the DATASET and FILE keywords; however, the data set name must be specified if the status of the data set is OLD or SHR, or if it is MOD and the data set currently exists. You will be prompted to supply the name of a MOD data set if you omit the SPACE operand, indicating that the data set currently exists. The SPACE operand must be specified when the data set is NEW.

The system generates names for SYSOUT data sets; therefore, you should not specify a data set name when you allocate a SYSOUT data set. If you do, the system ignores it.

FILE(name)
specifies the name to be associated with the data set. It may contain no more than eight characters. (This name corresponds to the Data Definition (DD) name in OS/360 Job Control Language and must match the DD name in the Data Control Block (DCB) that is associated with the data set.) For PL/I, this name is the file name in a DECLARE statement and has the form "DCL filename FILE"; for instance, DCL MASTER FILE. For COBOL, this name is the external-name used in the ASSIGN TO clause. For FORTRAN, this name is the data set reference number that identifies a data set and has the form "FTxxFyyy;" for instance, FT06F002.

If you omit this operand, the system assigns an available file name (ddname) from a data definition statement in the procedure that is invoked when you enter the LOGON command.

OLD
indicates that the data set currently exists and that you require exclusive use of the data set. The data set should be cataloged. If it is not, you must specify the VOLUME operand. OLD data sets are retained by the system when you free them from allocation.

SHR
indicates that the data set currently exists but that you do not require exclusive use of the data set. Other tasks may use it concurrently. SHR data sets are retained by the system when you free them.

MOD
indicates that you want to append data to the end of the data set. If the data set is actually new, you must also specify the SPACE operand. MOD data sets are retained by the system when you free them if you specify a data set name; they are deleted if you do not specify a data set name.

NEW
indicates that the data set does not exist and that it is to be created. You must specify the SPACE and BLOCK operands for NEW data sets. For new partitioned data sets you must also specify the DIR operand. NEW data sets are kept and cataloged if you specify a data set name. They are deleted if you do not specify a data set name.

SYSOUT
indicates that the data set is to be a system output data set. Output data will be initially written on a direct access device and later transcribed from the direct access device to the final output device. The final output device may be a unit record device (such as a printer or a terminal) or a magnetic tape device. The output class to which this data set is assigned is that of the message class. (See also the publication IBM System/360 Operating System, Supervisor and Data Management Services, GC28-6646.) After transcription by an output writer, SYSOUT data sets are deleted. You may specify space values with the SPACE operand; if you do not, default space values are provided by the system.

Note: If you do not specify OLD, SHR, MOD, NEW, or SYSOUT, the system assigns a default value depending on the BLOCK, SPACE, and DIR operands. If you specify the BLOCK and SPACE operands (for a sequential data set), or the BLOCK, SPACE and DIR operands (for a partitioned data set), the status defaults to NEW; otherwise, it defaults to OLD.

To change the output class refer to the FREE command and to the OUTPUT command.

VOLUME(serial)
specifies the serial number of the direct access volume on which a new data set is to reside or on which an old data set is located. If a volume is specified with an old data set, the data set must be on the volume or no allocation will take place. If you do not specify a serial number, new data sets are allocated to any eligible direct access volume, as determined by the UNIT information in your user entry in the UADS. If you do specify a volume serial number, eligibility is still determined by the UNIT information.

BLOCK(block-length)
specifies the average block length (in bytes) of the records that are to be written to the data set. The BLOCK operand is required for new data sets. You must specify the SPACE operand when you specify this operand. You may also specify BLOCK for SYSOUT data sets if the default values are not acceptable.

Note: The value supplied for BLOCK also becomes the value for BLKSIZE and is recorded in the DSCB for the data set unless you specify the USING operand. When you specify the USING operand, the BLKSIZE is taken from the attribute list.

SPACE(quantity,increment)
specifies the amount of space to be reserved for the new data set. The amount of space is determined by multiplying the "block length" (specified by the BLOCK(block-length) keyword) by the "quantity" value of the SPACE(quantity,increment) keyword. SPACE is required for new data sets and may be specified for SYSOUT data sets. You must specify the BLOCK operand when you specify this operand.

quantity
specifies the primary number of blocks to be allocated for the data set.

increment
specifies a secondary number of blocks to be allocated for the data set each time the previously allocated space has been exhausted. A maximum of 15 secondary blocks may be allocated.

DIR(integer)
specifies the number of 256 byte records that are to be allocated for the directory of a new partitioned data set. This operand must be specified if you are allocating a new partitioned data set. You must also specify the BLOCK and SPACE operands.

USING(attribute-list-name)
specifies the name of a list of attributes that you want to have assigned to the data set that you are allocating. The attributes in the list correspond to, and will be used for, data control block (DCB) parameters. (Note to user's familiar with conventional batch processing: these DCB parameters are the same as those normally specified by JCL and data management macro instructions.)

An attribute list must be stored in the system before you use this operand. You can build and name an attribute list by using the ATTRIB command. The name that you specify for the list when you use the ATTRIB command is the name that you must specify for this USING(attribute-list-name) operand.

Example 1

Operation: Allocate an existing cataloged data set containing input data for a program. The data set name conforms to the data set naming conventions, and you need exclusive use of the data.

Known: The name of the data set..... REB35.INPUT.DATA

```
ALLOCATE DATASET(INPUT.DATA) OLD
```

Example 2

Operation: Allocate a new data set to contain the output from a program.

Known: The name that you want to give the data set. REB35.OUTPUT.DATA
The block length..... 1056 bytes
The number of blocks expected to be used.... 50
DCB parameters are in an attribute list named ATTR.

```
ALLOCATE DATASET(OUTPUT.DATA) NEW SPACE(50,10) BLOCK(1056) USING(ATTR)
```

Example 3

Operation: Allocate your terminal as a temporary input data set.

```
ALLOCATE DATASET(*) FILE(FT01F001)
```

Example 4

Operation: Allocate an existing data set that is not cataloged and whose name does not conform to the data set naming conventions.

Known: The data set name..... SYS1.PTIMAC.AM
The volume serial number..... B99RS2
The DD name..... SYSLIB

```
alloc dataset('sys1.ptimac.am') file(syslib) volume(b99rs2) shr
```

Example 5

Operation: Allocate a new partitioned data set.

Known: The data set name..... JOHNS.OVERHEAD.TEXT
The block length..... 256 bytes
The number of blocks..... 500
The number of directory records..... 50

```
ALLOC DATASET(OVERHEAD.TEXT) NEW BLOCK(256) SPACE(500) DIR(50)
```

Use the ATTRIB command to build a list of attributes for data sets that you intend to allocate dynamically. During the remainder of your terminal session you can have the system refer to this list for data set attributes when you enter the ALLOCATE command. The ALLOCATE command will convert the attributes into DCB parameters for data sets being allocated.

Note: Before using this command, you should be familiar with DCB parameters as discussed in Job Control Language and Data Management Services.

COMMAND	OPERAND
{ATTRIB / ATTR}	attr-list-name [BLKSIZE(block-size)]
	[BUFL(buffer-length)]
	[BUFNO(number-of-buffers)]
	[KEYLEN(key-length)]
	[LRECL({logical-record-length / X})]
	[NCP(number-of-channel-programs)]
	[INPUT / OUTPUT]
	[EXPDT(year-day) / RETPD(number-of-days)]
	[BFALN({F / D})]
	[OPTCD(C,T,W and/or Q)]
	[EROPT({ACC / SKP / ABE})]
	[BFTEK({S / E / A / R})]
	[RECFM(A,B,F,M,S,T,U, and/or V)]

attr-list-name
specifies the name for the attribute list. This name can be specified later as a parameter of the ALLOCATE command. The name must consist of one through eight alphameric and/or national characters, must begin with an alphabetic or national character, and must be different from all other attr-list-names and ddnames that are in existence for your terminal session.

BLKSIZE(block-size)
specifies the blocksize for the data sets. The block size must be a decimal number and must not exceed 32,760 bytes.

The block size that you specify must be consistent with the requirements of the RECFM operand. If you specify:

- RECFM(F), then the block size must be equal to or greater than the logical record length.
- RECFM(F B), then the block size must be an integral multiple of the logical record length.
- RECFM(V), then the block size must be equal to or greater than the largest block in the data set. (Note: For unblocked variable-length records, the size of the largest block must allow space for the 4-byte block descriptor word in addition to the largest logical record length. The logical record length must allow space for a 4-byte record descriptor word. (See the publication IBM System/360 Operating System: Data Management Services, GC26-3746, for additional information.
- RECFM(V B), then the block size must be equal to or greater than the largest block in the data set. (Note: For blocked variable length records, the size of the largest block must allow space for the 4-byte block descriptor word in addition to the sum of the logical record lengths that will go into the block. Each logical record length must allow space for a 4-byte record descriptor word. Since the number of logical records can vary, you must estimate the optimum block size (and the average number of records for each block) based on your knowledge of the application that requires the I/O. (See the Data Management Services, publication for additional information.)

BUFL(buffer-length)
specifies the length, in bytes, of each buffer in the buffer pool. Substitute a decimal number for buffer-length. The number must not exceed 32,760.

If you omit this operand and the system acquires buffers automatically, the BLKSIZE and KEYLEN operands will be used to supply the information needed to establish buffer length.

BUFNO(number-of-buffers)
specifies the number of buffers to be assigned for data control blocks. Substitute a decimal number for number-of-buffers. The number must never exceed 255, and you may be limited to a smaller number of buffers depending on the limit established when the operating system was generated. The following table shows the condition that requires you to include this operand.

When you use one of the following methods of obtaining the buffer pool...	then:
(1) BUILD macro instruction	(1) you must specify BUFNO.
(2) GETPOOL macro instruction	(2) the system uses the number that you specify for GETPOOL.
(3) Automatically with BPAM or BSAM	(3) you must specify BUFNO.
(4) Automatically with QSAM	(4) you may omit BUFNO and accept two buffers.

KEYLEN(key-length)
specifies the length, in bytes, of each of the keys used to locate blocks of records in the data set when the data set resides on a direct access device.

The key-length must not exceed 255 bytes. If an existing data set has standard labels, you can omit this operand and let the system retrieve the key length from the standard label. If a key length is not supplied by any source before you issue an OPEN macro instruction, a length of zero (no keys) is assumed.

LRECL(logical-record-length)
specifies the length, in bytes, of the largest logical record in the data set. You must specify this operand for data sets that consist of either fixed length or variable length records.

Omit this operand if the data set contains undefined-length records.

The logical record length must be consistent with the requirements of the RECFM operand and must not exceed the block size (BLKSIZE operand) except for variable length spanned records. If you specify:

- RECFM(V) or RECFM(V B), then the logical record length is the sum of the length of the actual data fields plus 4 bytes for a record descriptor word.
- RECFM(F) or RECFM(F B), then the logical record length is the length of the actual data fields.
- RECFM(U), then you should omit the LRECL operand.

Note: For variable length spanned records (V S or V B S) processed by QSAM (locate mode) or BSAM, specify LRECL (X) when the logical record exceeds 32756 bytes.

NCP(number-of-channel-programs)
specifies the maximum number of READ or WRITE macro instructions allowed before a CHECK macro instruction is issued. The maximum number must not exceed 99 and must be less than 99 if a lower limit was established when the operating system was generated. If you are using chained scheduling, you must specify an NCP value greater than 1. If you omit the NCP operand, the default value is 1.

INPUT
specifies that the data set will be used only as input to a processing program.

OUTPUT
specifies that the data set will be used only to contain output from a processing program.

EXPDT(year-day)
specifies the data set expiration date. You must specify the year and day in the form "yyddd", where "yy" is a two digit decimal number for the year and "ddd" is a three digit decimal number for the day of the year. For example, January 1, 1974 is 74001 and December 31, 1975 is 75365.

RETPD(number-of-days)
specifies the data set retention period in days. The value must be a one through four digit decimal number.

BFALN({F|D})
specifies the boundary alignment of each buffer as follows:
F -- each buffer starts on a fullword boundary that is not a doubleword boundary.
D -- each buffer starts on a doubleword boundary.

If you do not specify this operand and it is not available from any other source, data management routines assign a doubleword boundary.

OPTCD(C,T,Q and/or W)
specifies the following optional services that you want the system to perform.
C -- You want to use chain scheduling.
T -- You want to use the user totaling facility.
W -- You want the system to perform a validity check when data is written on a direct access device.
Q -- You want to translate a magnetic tape from ASCII to EBCDIC or from EBCDIC to ASCII.

(You can request any or all of the services by combining the values for this operand. You may combine the characters in any sequence, being sure to separate them with blanks or commas).

EROPT({ACC|SKP|ABE})
specifies the option that you want executed if an error occurs when a record is read or written. The options are:
ACC -- accept the block of records in which the error was found.
SKP -- skip the block of records in which the error was found.
ABE -- end the task abnormally.

BFTEK({S|E|A|R})
specifies the type of buffering that you want the system to use. The types that you can specify are:

S -- simple buffering.
E -- exchange buffering.
A -- automatic record area buffering.
R -- record buffering.

RECFM(A,B,F,M,S,T,U, and/or V)
specifies the format and characteristics of the records in the data set. The format and characteristics must be completely described by one source only. If they are not available from any source, the default will be an undefined length record.

Use the following values with the RECFM operand.

A -- indicates that the record contains ASA printer control characters.
B -- indicates that the records are blocked.
F -- indicates that the records are of fixed length.
M -- indicates that the records contain machine code control characters.
S -- indicates that, for fixed-length records, the records are written as standard blocks (there must be no truncated blocks or unfilled tracks except for the last block or track). For variable length records, a record may span more than one block. Exhange buffering -BFTEK(E)- must not be used.
T -- indicates that the records may be written onto overflow tracks if required. Exchange buffering -BFTEK(E)- or chained scheduling -OPTCD(C)- cannot be used.
U -- indicates that the records are of undefined length.
V -- indicates that the records are of variable length.

You may specify one or more values for this operand (at least one is required). See the Job Control Language publication for a rigorous discussion of all possible valid combinations of values. The values must be separated by blanks or commas.

Example 1

Operation: Create a list of attributes to be assigned to a data set when the data set is allocated.

Known: The following attributes correspond to the DCB parameters that you want assigned to a data set.
Optional services: chain scheduling, user totaling.
Expiration date: Dec. 31, 1977.
Record format: variable length spanned records.
Error option: ABEND when READ or WRITE error occurs.
Buffering: simple buffering.
Boundary alignment: doubleword boundary.
Logical record length: records may be larger than 32,765 bytes.
The name for this attribute list is DCBPARMS.

```
ATTR  DCBPARMS  OPTCD(C T)  EXPDT(77365)  RECFM(V S) -
EROPT(ABE)   BFTEK(S)   BFALN(D)  LRECL(X)
```

Example 2

Operation: This example shows how to create an attribute list, how to use the list when allocating two data sets, and how to delete the list so that it cannot be used again.

Known: The name for the attribute list..... DSATTRS
The attributes..... EXPDT(99365) BLKSIZE(24000) BFTEK(A)
The name for the first data set..... FORMAT.INPUT
The name of the second data set..... TRAJECT.INPUT

```
attrib  dsattrs  expdt(99365)  blksize(24000)  bftek(a)

allocate  dataset(format.input)  new block(80)  space(1,1) -
volume(111111)  using(dsattrs)

alloc  da(traject.input)  old bl(80)  volume(111111)  using(dsattrs)

free  attrlist(dsattrs)
```

CALL Command

Use the CALL command to load and execute a program that exists in executable (load module) form. The program may be user-written, or it may be a system module such as a compiler, sort, or utility program.

You must specify the name of the program (load module) to be processed. It must be a member of a partitioned data set.

You may specify a list of parameters to be passed to the specified program. The system formats this data so that when the program receives control, register one contains the address of a fullword. The three low order bytes of this fullword contain the address of a halfword field. This halfword field is the count of the number of bytes of information contained in the parameter list. The parameters immediately follow the halfword field.

If the program terminates abnormally, you are notified of the condition and may enter a TEST command to examine the failing program.

COMMAND	OPERANDS
CALL	data-set-name ['parameter-string']

data-set-name
: specifies the name of the member of a partitioned data set that contains the program to be executed. You must enclose the member name within parentheses. When the name of the partitioned data set conforms to the data set naming conventions, the system will add the necessary qualifiers to make the name fully qualified. The system will supply .LOAD as a default for the descriptive qualifier and (TEMPNAME) as the default for a member name. If the name of the partitioned data set does not conform to the data set naming conventions, it must be included with the member name in the following manner:

```
data-set-name(membername)
```

If you specify a fully qualified name, enclose it in apostrophes (single quotes) in the following manner:

```
'USERID.MYPROGS.LOADMOD(A)'
'SYS1.LINKLIB(IEUASM)'
```

parameter string
: specifies up to 100 characters of information that you want to pass to the program as a parameter list. When passing parameters to a program, you should use the standard linkage conventions.

Example 1

Operation: Execute a load module.

Known: The name of the load module..... BARB01.PEARL.LOAD(TEMPNAME)
Parameters..... 10,18,23

```
CALL PEARL '10 18 23'
```

Example 2

Operation: Execute a load module.

Known: The name of the load module..... SHEP.MYLIB.LOAD(COS1)

```
CALL MYLIB(COS1)
```

Example 3

Operation: Execute a load module.

Known: The name of the load module..... BCMD93.LOAD(SIN1)

```
CALL (SIN1)
```

Use the CANCEL command to halt processing of conventional batch jobs that you have submitted from your terminal. If several jobs have the same jobname, the system cancels only the first one it finds with that name. A message will be displayed at your terminal to advise you of the action taken by the system. A message will also be displayed at the system operator's console when a job is canceled.

Only authorized users can use this command (see the ACCOUNT command). This command is generally used in conjunction with the SUBMIT, STATUS, and OUTPUT commands.

COMMAND	OPERANDS
CANCEL	(job-name-list)

(job-name-list)
specifies the names of the jobs that you want to cancel. The name of a job that you submit from your terminal should consist of your user identification plus one or more alphameric characters up to a maximum of eight characters. You can only cancel jobs that have a userid that is identical to the one with which you logged on.

Note: When you specify a list of several job names, you must separate the jobnames with standard delimiters and you must enclose the entire list within parentheses.

Example 1

Operation: Cancel a conventional batch job.

Known: The name of the job..... JE024A1

```
CANCEL JE024A1
```

Example 2

Operation: Cancel several conventional batch jobs.

Known: The names of the jobs..... D58BOBTA
D58BOBTB
D58BOBTC

```
CANCEL (D58BOBTA D58BOBTB D58BOBTC)
```

DELETE Command

Use the DELETE command to delete one or more data sets or one or more members of a partitioned data set.

If the data set is cataloged, the system removes the catalog entry. The catalog entry for a partitioned data set is removed only when the entire partitioned data set is deleted. The system deletes a member of a partitioned data set by removing the member name from the directory of the partitioned data set.

Members of a partitioned data set and aliases for any members must each be deleted explicitly. That is, when you delete a member, the system does not remove any alias names of the member; likewise, when you delete an alias name, the member itself is not deleted.

After you delete a protected data set, you should use the PROTECT command to update the password data set to reflect the change. This will prevent your having insufficient space for future entries.

COMMAND	OPERANDS
{DELETE / D}	(data-set-list) [PURGE / NOPURGE]

data-set-list
: specifies the name of a data set or a member of a partitioned data set, or a list of names of data sets and/or members (see data set naming conventions). If you specify a list, it must be enclosed within parentheses.

  If you want to delete several data sets having similar names, you may insert an asterisk into the data set name at the point of dissimilarity. That is, all data sets whose names match except at the position where the asterisk is placed will be deleted. However, you may use only one asterisk per data set name, and you must <u>not</u> place it in the first position.

  For instance, suppose that you have several data sets named:

```
ROGERA.SOURCE.PLI
ROGERA.SOURCE2.PLI
ROGERA.SOURCE2.TEXT
ROGERA.SOURCE2.DATA
```

  If you specify:

```
DELETE SOURCE2.*
```

  the only data set remaining will be

```
ROGERA.SOURCE.PLI
```

PURGE
: specifies that the data set is to be deleted even if its expiration date has not elapsed. This operand is ignored by the system if you are deleting a member of a partitioned data set. The PURGE keyword applies to all data sets specified in a list.

NOPURGE
specifies that you want the system to check the expiration date for the data set. Only if the expiration date has elapsed will the data set be deleted. The NOPURGE keyword applies to all data sets specified in a list. This is the default if neither PURGE nor NOPURGE is specified.

Example 1

Operation: Delete a member of a partitioned data set.

Known: The data set name and member name..... BAN00.INCREASE.FORT(HOOF)

```
DELETE INCREASE.FORT(HOOF)
```

Example 2

Operation: Delete several data sets.

Known: The name of the data sets..... JWSD58.CMDS.TEXT
JWSD58.UTILS.OBJ
JWSD58.BUDGET.ASM

```
DELETE (CMDS.TEXT UTILS.OBJ BUDGET.ASM)
```

Example 3

Operation: Delete a data set even if its expiration date has not expired.

Known: The name of the data set REB1.SCHEDULE.OBJ

```
D SCHEDULE.OBJ PURGE
```

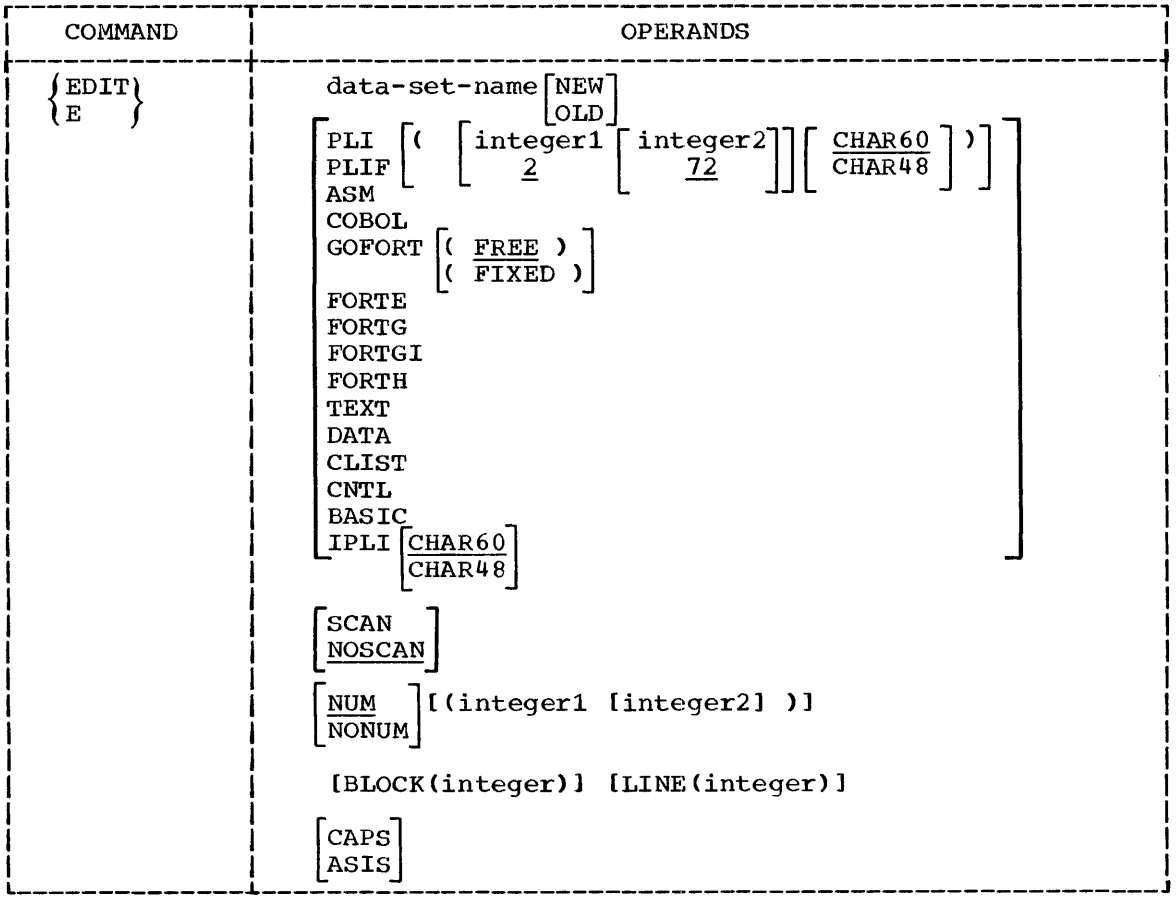
The EDIT command is the primary facility for entering data into the system. Therefore, almost every application involves some use of EDIT. With EDIT and its subcommands, you can create, modify, store, and retrieve data sets with sequential or partitioned data set organization. These data sets may contain:

- Source programs composed of programming language statements (PL/1, COBOL, FORTRAN, etc.)
- Data used as input to a program.
- Text used for information storage and retrieval.
- Commands, subcommands, and/or data (Command Procedure).
- Job Control (JCL) statements for background jobs.

The EDIT command will only support data sets that have one of the following record formats:

F - fixed length records
FB - fixed length, blocked records
V - variable length records
VB - variable length, blocked records

See Figure 8, note 4 for a complete list of the data set attributes that are supported by EDIT.



data-set-name
specifies the name of the data set that you want to create or edit. (See data set naming conventions.)

Note: Any user-defined data set type (specified at system generation) is also a valid data set type keyword and may have subfield parameters defined by the user's installation (see Figure 8, note 4).

PLI
specifies that the data set identified by the first operand is for PL/I statements. The statements may be for the PL/I Optimizing Compiler or the PL/I Checkout Compiler.

PLIF
specifies that the data set specified by the first operand is for statements for the PL/I(F) Compiler.

integer1 and integer2
the optional values contained within the parentheses are applicable only when you request syntax checking. The integer1 and integer2 values define the column boundaries for your input statements. The position of the first character of a line, as determined by the left margin adjustment on your terminal, is column 1. The value for integer1 specifies the column where each input statement is to begin. The statement can extend from the column specified by integer1 up to and including the column specified as a value for integer2. If you omit integer1 you must omit integer2, and the default values are columns 2 and 72; however, you can omit integer2 without omitting integer1.

CHAR48 or CHAR60
CHAR48 specifies that the PL/I source statements are written using the character set that consists of 48 characters. CHAR60 specifies that the source statements are written using the character set that consists of 60 characters. If you omit both CHAR48 and CHAR60, the default value is CHAR60.

IPLI(CHAR48 or CHAR60)
specifies that the data set identified by the first operand is for PL/I statements that may be processed by the ITF:PLI Program Product. CHAR48 or CHAR60 are used as described in the PLI operand description.

BASIC
specifies that the data set identified by the first operand is for BASIC statements that may be processed by the ITF:BASIC Program Product.

ASM
specifies that the data set identified by the first operand is for assembler language statements.

COBOL
specifies that the data set identified by the first operand is for COBOL statements.

CLIST
specifies that the data set identified by the first operand is for a command procedure and will contain TSO commands and subcommands as statements or records in the data set.

CNTL
specifies that the data set identified by the first operand is for Job Control Language (JCL) statements and SYSIN data to be used with the SUBMIT command.

TEXT
specifies that the data set identified by the first operand is for text that may consist of both uppercase and lowercase characters.

DATA
specifies that the data set identified by the first operand is for data that may be subsequently retrieved or used as input data for processing by an application program.

FORTE
specifies that the data set identified by the first operand is for FORTRAN (E) statements.

FORTG
specifies that the data set identified by the first operand is for FORTRAN (G) statements.

FORTGI
specifies that the data set identified by the first operand is for FORTRAN (G1) statements. You may use FORT as an abbreviation for this operand. This is the default value if no other FORTRAN language level is specified with the FORT operand.

FORTH
specifies that the data set identified by the first operand is for FORTRAN (H) statements.

GOFORT(FREE or FIXED)
specifies that the data set identified by the first operand is for statements that are suitable for processing by the Code and Go FORTRAN Program Product.

FREE specifies that the statements are of variable lengths and do not conform to set column requirements. This is the default value if neither FREE nor FIXED is specified. FIXED specifies that statements adhere to standard FORTRAN column requirements and are 80 bytes long.

Note: The ASM, BASIC, CLIST, CNTL, COBOL, DATA, FORTE, FORTG, FORTGI, FORTH, GOFORT, IPLI, PLI and TEXT operands specify the type of data set you want to edit or create. You must specify one of these whenever:

- The data-set-name operand does not follow data set naming conventions (i.e., it is enclosed in quotes).
- The data-set-name operand is a member name only (i.e., it is enclosed in parentheses).
- The data-set-name operand does not include a descriptive qualifier; or the descriptive qualifier is such that EDIT cannot determine the data set type. (See Figure 3 for a list of valid descriptive qualifiers.)

The system prompts the user for data set type whenever the type cannot be determined from the descriptive qualifier (as in the 3 cases above), or whenever the user forgets to specify a descriptive qualifier on the EDIT command.

Note: When the descriptive qualifier FORT is entered with no data set type, the data set type default is GOFORT(FREE). If PLI is the descriptive qualifier, the data set type default is PLI. To use data set types GOFORT(FIXED), FORTGI, FORTG, FORTE, FORTH or PLIF, you must enter the data set type keyword.

NEW
specifies that the data set named by the first operand does not exist. If an existing cataloged data set already has the data set name that you specified, the system notifies you when you try to save it; otherwise, the system allocates your data set when you save it.

If you specify NEW without specifying a member name, the system allocates a sequential data set for you when you save it. If you specify NEW and include a member name the system allocates a partitioned data set and creates the indicated member when you try to save it.

OLD
specifies that the data set named on the EDIT command already exists. When you specify OLD and the system is unable to locate the data set, you will be notified and you will have to reenter the EDIT command.

If you specify OLD without specifying a member name, the system will assume that your data set is sequential: if the data set is in fact a partitioned data set, the system will assume that the member name is TEMPNAME. If you specify OLD and include a member name, the system will notify you if your data set is not partitioned.

If you do not specify OLD or NEW, the system uses a tentative default of OLD. If the data set name or member name that you specified, cannot be located, you will be prompted to enter NEW or OLD. If you enter NEW, EDIT processing will continue. If you enter OLD, the system will notify you why the data set or member could not be located. You can then enter EDIT or another command.

SCAN
specifies that each line of data you enter in Input mode is to be checked statement by statement for proper syntax. If you specify the BASIC or IPLI data set type keyword, all modifications made in Edit mode and each line of data entered in Input mode will be checked for proper syntax. Syntax checking is available only for statements written in GOFORT, FORTE, FORTGI, FORTG, FORTH, BASIC, IPLI, PLI, or PLIF.

Note: User-defined data set types can also use this keyword if a syntax checker name was specified at system generation time.

NOSCAN
specifies that syntax checking is not to be performed. This is the default value if neither SCAN nor NOSCAN is specified.

NUM(integer1 integer2)
specifies that the lines of the data set records are numbered. You may specify integer1 and integer2 for ASM type data sets only. Integer1 specifies, in decimal, the starting column (73-80) of the line number. Integer2 specifies, in decimal, the length (8 or less) of the line number. Integer1 plus integer2 cannot exceed 81. If integer1 and integer2 are not specified, the line numbers will default according to the type of data set being created or edited (see Table 4). NUM is the default value if you omit both NUM and NONUM.

NONUM
specifies that your data set records do not contain line numbers. Do not specify this keyword for the BASIC, IPLI, and GOFORT data set types, since they must always have line numbers. The default is NUM.

CAPS
specifies that all input data is to be converted to uppercase characters. If you omit both CAPS and ASIS, then CAPS is the default except when the data set type is TEXT.

ASIS
specifies that input is to retain the same form (upper and lower case) as entered. ASIS is the default for TEXT only.

BLOCK(integer)
specifies the maximum length, in bytes, for blocks of records of a new data set. Specify this operand only when creating a new data set. You cannot change the block size of an existing data set. If you omit this operand, it will default according to the type of data set being created. Default block sizes are described in Table 4. If different defaults are established at system generation (SYSGEN) time, Table 4 values may not be applicable. The blocksize (BLOCK) for data sets that contain fixed length records must be a multiple of the record length (LINE); for variable length records, the blocksize must be a multiple of the record length plus 4.

LINE(integer)
specifies the length of the records to be created for a new data set. Specify this operand only when creating a new data set. The new data set will be composed of fixed length records with a logical record length equal to the specified integer. You cannot change the logical record size of an existing data set. If you specify this operand and the data set type is ASM, FORTE, FORTG, FORTGI, FORTH, GOFORT(FIXED), COBOL or CNTL the integer must be 80. If this operand is omitted, the line size defaults according to the type of data set being created. Default line sizes for each data set type may be found in Figure 8. This operand is used in conjunction with the BLOCK operand.

DATA SET TYPE	DSORG	LRECL		BLOCK SIZE		LINE NUMBERS		
		LINE(n)		(Note1) BLOCK(n)		NUM (n,m)	CAPS/ASIS	
		default	specif.	default	specif.	default (n,m) spec.	default	CAPS Required
ASM	PS/PO	80	=80	1680	≤default	Last 8 73≤n≤80	CAPS	Yes
BASIC	PS/PO	120	(Note 2)	1680	≤default	(Note 3)	CAPS	Yes
CLIST	PS/PO	255	(Note 2)	1680	≤default	(Note 3)	CAPS	Yes
CNTL	PS/PO	80	=80	1680	≤default	Last 8	CAPS	Yes
COBOL	PS/PO	80	=80	400	≤default	First 6	CAPS	Yes
DATA	PS/PO	80	≤255	1680	≤default	Last 8	CAPS	No
FORTE, FORTG, FORTGI, FORTH, GOFORT								
(FIXED) GOFORT	PS/PO	80	=80	400	≤default	Last 8	CAPS	Yes
(FREE)	PS/PO	255		1680	≤default	First 8	CAPS	No
IPLI	PS/PO	120	(Note 2)	1680	≤default	(Note 3)	CAPS	Yes
(or user supplied data set type -- See Note 4)								
PLI	PS/PO	104	≤100	500	≤default	(Note 3)	CAPS	No
PLIF	PS/PO	80	≤100	400	≤default	Last 8	CAPS	Yes
TEXT	PS/PO	255	(Note 2)	1680	≤default	(Note 3)	ASIS	No

Note 1:
The default or maximum allowable block size may be specified at SYSGEN time.

Note 2:
Specifying a LINE value results in fixed length records with a LRECL equal to the specified value. The specified value must always be equal to or less than the default. If the LINE keyword is omitted, variable length records will be created.

Note 3:
The line numbers will be contained in the last eight bytes of all fixed length records and in the first eight bytes of all variable length records.

Note 4:
A user can have additional data set types recognized by the EDIT command processor. These user-defined data set types, along with any of the data set types shown above, can be defined at system generation time by using the EDIT macro. The EDIT macro causes a table of constants to be built which describes the data set attributes. For more information on how to specify the EDIT macro at system generation time, refer to the publication, IBM System/360 Operating System: System Generation, GC28-6554.

When a user wants to edit a data set type that he has defined himself, the data set type is used as the descriptor (rightmost) qualifier. The user cannot override any data set types that have been defined by IBM. The EDIT command processor will support data sets that have the following attributes:

```
Data Set Organization- Must be either sequential or partitioned
Record formats-        Fixed or Variable
Logical Record Size-   Less than or equal to 255 characters
Block Sizes-           User specified -- must be less than or equal to track length
Sequence Nos.-         V type:  First 8 characters
                       F type:  Last 8 characters
```

Figure 8. Default Values for LINE and BLOCK Operands

You can also use the EDIT command to:

- Compile, load, and execute a source program.

These operations are defined and controlled by using the EDIT operands and subcommands.

Modes of Operation

The EDIT command has two modes of operation: input mode and edit mode. You enter data into a data set when you are in input mode. You enter subcommands and their operands when you are in edit mode.

You must specify a data set name when you enter the EDIT command. If you specify the NEW keyword, the system places you in the Input mode. If you do not specify the NEW keyword, you are placed in the Edit mode if your specified data set is not empty; if the data set is empty, you will be placed in Input mode.

You can limit access to your data set by specifying a password when you use the EDIT command. To specify a password, enter a slash (/) followed by the password of your choice after the data set name operand of the EDIT command.

Input Mode

In input mode, you type a line of data and then enter it into the data set by pressing your terminal's carrier return key. You can enter lines of data as long as you are in input mode. One typed line of input becomes one record in the data set.

CAUTION: If you enter a command or subcommand while you are in input mode, the system will add it to the data set as input data. Enter a null-line to return to EDIT mode before entering any subcommands.

Line Numbers: Unless you specify otherwise, the system assigns a line number to each line as it is entered. Line numbers make editing much easier, since you can refer to each line by its own number.

Each line number consists of not more than eight digits, with the significant digits justified on the right and preceded by zeros. Line numbers are placed at the beginning of variable length records and at the end of fixed length records (exception: line numbers for COBOL fixed length records are placed in the first six positions at the beginning of the record). When you are working with a data set that has line numbers, you can have the new line number listed at the start of each new input line. If you are creating a data set without line numbers, you can request that a prompting character be displayed at the terminal before each line is entered. Otherwise, none will be issued.

Record Format: Record formats and sizes may vary according to the type of data set. In all cases, the length of your records must not exceed 255 characters, and the record format cannot be other than fixed (F), fixed blocked (FB), variable (V), or variable blocked (VB).

Note: Edit does not allow a user to edit data sets with record formats of either FBA or FBM.

All input records will be converted to upper case characters, except when you specify the ASIS or TEXT operand. The TEXT operand also specifies that character-deleting indicators will be recognized, but all other characters will be added to the data set unchanged. More specific considerations are:

All Assembler source data sets must consist of fixed length records 80 characters in length. These records may or may not have line numbers. If the records are line-numbered, the number can be located anywhere within columns 73 to 80 of the stored record (the printed line number always appears at the left margin).

IPLI and BASIC data sets may consist of either fixed length or variable length records. All records must contain line numbers. Fixed-length records may be specified up to 120 characters in length. The default is variable-length records with the line number contained in the first eight characters.

You can create a variety of FORTRAN data sets: FORTE; FORTG; FORTGI; FORTH; and GOFORT. You can enter GOFORT input statements in "free form", that is, there are no specific columns into which your data must go. Free form FORTRAN data will be stored in variable length records.

Syntax Checking: You can have each line of input checked for proper syntax. The system will check the syntax of statements for data sets having FORT, PLI, IPLI, and BASIC descriptive qualifiers. Input lines will be collected within the system until a complete statement is available for checking.

When an error is found during syntax checking, an appropriate error message is issued and edit mode is entered. You can then take corrective action, using the subcommands. When you wish to resume input operations, press your terminal's carrier return key without typing any input. Input mode is then entered and you can continue where you left off. Whenever statements are being checked for syntax during input mode, the system will prompt you for each line to be entered unless you specify the NOPROMPT operand for the INPUT subcommand.

Continuation of a Line in Input Mode: In input mode there are three independent situations that require you to indicate the continuation of a line - by ending it with a hyphen - (i.e., a hyphen followed immediately by a carriage return). The situations are:

1. The syntax checking facility is being used.
2. The data set type is GOFORT(FREE).
3. The data set type is CLIST (variable length records).

If none of these situations apply, avoid ending a line with a hyphen (minus sign) since it will be removed by the system before storing the line in your data set.

You must use the hyphen when the syntax checking facility is active to indicate that the logical line to be syntax checked consists of multiple input lines. The editor will then collect these lines (removing the hyphens) and pass them as one logical line to the syntax scanner. However, each individual input line (with its hyphen removed) is also stored separately in your data set.

You must use the hyphen to indicate logical line continuation in a GOFORT(FREE) data set, whether or not syntax checking is active. Since the Code and Go Fortran Free-form input format requires a hyphen to indicate continuation to its syntax checker and compiler, the hyphen is not removed from the input line by EDIT, but becomes part of the stored line in your data set.

The hyphen is also used to indicate logical line continuation in command procedures (CLIST data sets). If the CLIST is in variable length record format (the default), the hyphen is not removed by EDIT but becomes part of the stored line in your data set and will be recognized when executed by the EXEC command processor. If the CLIST is in fixed length record format, a hyphen, placed eight character positions before the end of the record and followed by a blank, will be recognized as a continuation when executed by the EXEC command processor. (This assumes that the line number field is defined to occupy the last eight positions of the stored record.) For example, if the parameter LINE(80) was specified on the EDIT command when defining the CLIST data set, the hyphen must be placed in data position 72 of the input line followed immediately by a blank. (Location of a particular input data column is described under the TABSET subcommand of EDIT.)

Note that these rules apply only when entering data in Input Mode. When you use a subcommand (e.g., CHANGE, INSERT) to enter data, a hyphen at the end of the line indicates subcommand continuation; the system will append the continuation data to the subcommand.

To insert a line of data ending in a hyphen in situations where the system would remove the hyphen (i.e., while in subcommand mode or in input mode for other than CLIST or GOFORT data sets), enter a hyphen in the next-to-last column, a blank in the last column, and an immediate carriage return.

Edit Mode

You can enter subcommands to edit data sets when you are in Edit Mode. You can edit data sets that have line numbers by referring to the number of the line that you want to edit. This is called line-number editing. You can also edit data by referring to specific items of text within the lines. This is called context editing. A data set having no line numbers may be edited only by context. Context editing is performed by using subcommands that refer to the current line value or a character combination, such as with the FIND or CHANGE subcommands. There is a pointer within the system that points to a line within the data set. Normally, this pointer points to the last line that you referred to. You can use subcommands to change the pointer so that it points to any line of data that you choose. You may then refer to the line that it points to by specifying an asterisk (*) instead of a line number. Figure 9 shows where the pointer points at completion of each subcommand.

Note: A current-line pointer value of zero refers to the position before the first record, if the data set does not contain a record zero.

When you edit data sets with line numbers, the line number field will not be involved in any modifications made to the record except during renumbering. Also, the only editing operations that will be performed across record boundaries will be the CHANGE and FIND subcommands, when the TEXT and NONUM operands have been specified for the EDIT command. In CHANGE and FIND an editing operation will be performed across only one record boundary at a time.

Edit Subcommands	Value of the Pointer at Completion of Subcommand
BOTTOM	Last line (or zero for empty data sets)
CHANGE	Last line changed
DELETE	Line preceding deleted line (or zero if the first line of the data set has been deleted)
DOWN	Line n relative lines below the last line referred to, where n is the value of the "count" parameter, or bottom of the data set (or line zero for empty data sets)
END	No change
FIND	Line containing specified string, if any; else, no change
FORMAT(a program product)	No change
HELP	No change
INPUT	Last line entered
INSERT	Last line entered
Insert/Replace/Delete	Inserted line or replaced line or line preceding the deleted line if any (or zero, if no preceding line exists)
LIST	Last line listed
MERGE(a program product)	Last line
PROFILE	No change
RENUM	Same relative line
RUN	No change
SAVE	No change
SCAN	Last line scanned, if any
TABSET	No change
TOP	Zero value
UP	Line n relative lines above the last line referred to, where n is the value of the "count" parameter, (or line zero for empty data sets).
VERIFY	No change

Figure 9. Values of the Line Pointer Referred to by an Asterisk (*)

Changing From One Mode to Another

If you specify an existing data set name as an operand for the EDIT command, you begin processing in edit mode. If you specify a new data set name or an old data set with no records, as an operand for the EDIT command, you will begin processing in input mode. You will change from edit mode to input mode when:

1. You press the carriage return key without typing anything first.

Note: If this is the first time during your current usage of EDIT that Input mode is entered, input will begin at the line after the last line of the data set (for data sets which are not empty) or at the first line of the data set (for empty data sets). If this is not the first time during your current usage of EDIT that Input mode is entered, input will begin at the point following the data entered when last in input mode.

2. You enter the INPUT subcommand.

Note: If you use the INPUT subcommand without the R keyword and the line is null (that is, it contains no data), input begins at the specified line; if the specified line contains data, input begins at the first increment past that line. If you use the INPUT subcommand with the R keyword, input begins at the specified line, replacing existing data, if any.

3. You enter the INSERT subcommand with no operands.

You will switch from input mode to edit mode when:

1. You press the carriage return key without typing anything first.

2. You cause an attention interruption.

3. There is no more space for records to be inserted into the data set and resequencing is not allowed.

4. When an error is discovered by the syntax checker.

Data Set Disposition

The system assumes a disposition of (NEW,CATLG) for new data sets and (OLD,KEEP) for existing data sets.

TABULATION CHARACTERS

When you enter the EDIT command into the system, the system establishes a list of tab setting values for you, depending on the data set type. These are logical tab setting values and may or may not represent the actual tab setting on your terminal. You can establish your own tab settings for input by using the TABSET subcommand. A list of the default tab setting values for each data set type is presented in the TABSET subcommand description. The system will scan each input line for tabulation characters (the characters produced by pressing the TAB key on the terminal). The system will replace each tabulation character by as many blanks as are necessary to position the next character at the appropriate logical tab setting.

When tab settings are not in use, each tabulation character encountered in all input data will be replaced by a single blank. You can also use the tabulation character to separate subcommands from their operands.

EXECUTING USER WRITTEN PROGRAMS

You can compile and execute the source statements contained in certain data set types by using the RUN subcommand. The RUN subcommand makes use of optional Program Products; the specific requirements are discussed in the description of the RUN subcommand.

TERMINATING THE EDIT COMMAND

You can terminate the EDIT operation at any time by switching to edit mode (if you are not already in edit mode) and entering the END subcommand. Before terminating the EDIT command, you should be sure to store all data that you want to save. You can use the SAVE subcommand for this purpose.

Example 1

Operation: Create a data set to contain a COBOL program.

Known: The user-supplied name for the new data set. PARTS
The fully qualified name will be..... BOBD58.PARTS.COBOL
Line numbers are to be assigned.

```
EDIT PARTS NEW COBOL
```

Example 2

Operation: Create a data set to contain a program written in FORTRAN to be processed by the FORTRAN (G1) compiler.

Known: The user-supplied name for the new data set..... HYDRLICS
The fully qualified name will be..... DEPT90.HYDRLICS.FORT
The input statements are not to be numbered.
Syntax checking is desired.
Block size..... 400
Line length must be..... 80
The data is to be changed to all upper case.

```
EDIT HYDRLICS NEW FORTGI NONUM SCAN
```

or

```
e hydrlics new fortgi scan nonum
```

Example 3

Operation: Add data to an existing data set containing input data for a program.

Known: The name of the data set..... FHETD58.MANHRS.DATA
Block size..... 1680
Line length..... 80
Line numbers are desired.
The data is to be upper case.
Syntax checking is not applicable.

```
e manhrs.data
```

Example 4

Operation: Create a data set for a Command Procedure.

Known: The user supplied name for the data set..... CMDPROC

```
E CMDPROC NEW CLIST
```

Example 5

Operation: Create a data set to contain a PLI PROGRAM.

Known: The user-supplied name for the data set..... WEATHER
The column requirements for input records
left margin..... Column 1
right margin..... Column 68
The allowed character set..... 48 characters
Line numbers are desired.
Each statement is to be checked for proper syntax.
The default BLOCK and LINE value are acceptable.

```
edit weather new pli(1 68 char48)  scan
```

SUBCOMMANDS FOR EDIT

Use the subcommands while in edit mode to edit and manipulate data. The format of each subcommand is similar to the format of all the commands. Each subcommand, therefore, is presented and explained in a manner similar to that for a command. Figure 10 contains a brief summary of each subcommand's function.

BOTTOM	Moves the pointer to the last line.
CHANGE	Modifies text of a line, or range of lines.
DELETE	Removes records.
DOWN	Moves the pointer toward the end of the data.
END	Terminates the EDIT command.
FIND	Locates a character string.
FORMAT (available as an optional Program Product)	Formats and lists data.
HELP	Explains available subcommands.
INPUT	Prepares the system for data input.
INSERT	Inserts records.
Insert/Replace/Delete	Inserts,replaces, or deletes a line.
LIST	Prints out specific lines of data.
MERGE (available as an optional Program Product)	Combines all or parts of data sets.
PROFILE	Specifies your selected 'delete' indicators.
RENUM	Numbers or renumbers lines of data.
RUN	Causes compilation and execution of data set.
SAVE	Retains the data set.
SCAN	Controls syntax checking.
TABSET	Sets the Tabs.
TOP	Sets the pointer to zero value.
UP	Moves the pointer toward the start of data set.
VERIFY	Causes current line to be listed whenever the current line pointer changes or the text of the current line is modified.

Figure 10. Subcommands Used With the Edit Command

Use the BOTTOM subcommand to change the current line pointer so that it points to the last line of the data set being edited or so that it contains a zero value, if the data set is empty. This subcommand may be useful when subsequent subcommands such as INPUT or MERGE must begin at the end of the data set.

SUBCOMMAND	OPERANDS
{BOTTOM {B	

Use the CHANGE subcommand to modify a sequence of characters (a character-string) in a line or in a range of lines. Either the first occurrence or all occurrences of the sequence can be modified.

SUBCOMMAND	OPERANDS
{CHANGE / C}	[* / line-number-1 [line-number-2] / *[count 1]] {string1 [string2 [special-delimiter[ALL]]] / count2}

line-number-1
specifies the number of a line you want to change. When used with line-number-2, it specifies the first line of a range of lines.

line-number-2
specifies the last line of a range of lines that you want to change. The specified lines are scanned for occurrences of the sequence of characters specified for string1. If you specify the ALL operand, each occurrence of string1 in the range of lines is replaced by the sequence of characters that you specify for string2. If you do not specify the ALL operand, only the first occurrence of string1 will be replaced by string2.

*
specifies that the line pointed to by the line pointer in the system is to be used. If you do not specify a line number or an asterisk, the current line will be the default value.

count1
specifies the number of lines that you want to change, starting at the position indicated by the asterisk (*).

string1
specifies a sequence of characters (a character string) that you want to change. The sequence must be (1) enclosed within single quotes, or (2) preceded by an extra character which serves as a special delimiter. The extra character may be any printable character other than a single quote (apostrophe), number, blank, tab, comma, semicolon, parenthesis, or asterisk. The hyphen (-) can be used but should be avoided due to possible confusion with its use in continuation. The extra character must not appear in the character string. Do not put a standard delimiter between the extra character and the string of characters unless you intend for the delimiter to be treated as a character in the character string.

If string1 is specified and string2 is not, the specified characters are displayed at your terminal up to (but not including) the sequence of characters that you specified for string1. You can then edit the sequence of characters as you please.

string2
specifies a sequence of characters that you want to use as a replacement for string1. Like string1, string2 must be (1) enclosed within single quotes, or (2) preceded by a special delimiter. This delimeter must be the same as the extra character used for string1.

ALL
specifies that every occurrence of string1 within the specified line or range of lines will be replaced by string2. If this operand is omitted, only the first occurrence of string1 will be replaced with string2.

Note: If the special delimeter form is used, string2 must be terminated by the delimeter before typing the ALL operand.

count2
specifies a number of characters to be displayed at your terminal, starting at the beginning of each specified line.

Quoted String Notation

As indicated above, instead of using special delimiters to indicate a character string, you can use paired single quotes (apostrophes) to accomplish the same function with the CHANGE subcommand. The use of single quotes as delimiters for a character string is called quoted-string notation. Following are the rules for quoted-string notation for the string1 and string2 operands:

1. You cannot use both special-delimiter and quoted-string notation in the same subcommand.
2. Each string must be enclosed within single quotes, e.g., 'This is string1' 'This is string2.'
3. A single quote within a character string is represented by two single quotes, e.g., 'pilgrim''s progress'.
4. A null string is represented by two single quotes, e.g., ''.

You can specify quoted string notation in place of special delimiter-notation to accomplish any of the functions of the CHANGE subcommand as follows:

Function	*Special Delimiter Notation	Quoted String Notation
Replace	+ab+cde+	'ab' 'cde'
Delete	+ab++	'ab' ''
Print up to	+ab	'ab'
Place in front of	++cde+	'' 'cde'

* - using the + sign as the delimeter.

Note: It is recommended that you choose the form that best suits you (either special delimeter or quoted string) and use it consistently. It will expedite your use of this powerful subcommand.

Combinations of Operands

You can enter several different combinations of these operands. The system interprets the operands that you enter according to the following rules:

- When you enter a single number and no other operands, the system assumes that you are accepting the default value of the asterisk (*) and that the number is a value for the count2 operand.
- When you enter two numbers and no other operands, the system assumes that they are line-number-1 and count2 respectively.
- When you enter two operands and the first is a number and the second begins with a character that is not a number, the system assumes that they are line-number-1 and string1.
- When you enter three operands and they are all numbers, the system assumes that they are line-number-1, line-number-2 and count2.
- When you enter three operands and the first two are numbers but the last begins with a character that is not a number, the system assumes that they are line-number-1, line-number-2 and string1.

Example 1

Operation: Change a sequence of characters in a particular line of a line numbered data set.

Known: The line number..... 57
The old sequence of characters..... parameter
The new sequence of characters..... operand

```
CHANGE 57 XparameterXoperand
```

Example 2

Operation: Change a sequence of characters wherever it appears in several lines of a line numbered data set.

Known: The starting line number..... 24
The ending line number..... 82
The old sequence of characters..... i.e.
The new sequence of characters..... e.g.

```
change 24 82 !i.e. !e.g. !all
```

The blanks following the string1 and string2 examples (i.e. and e.g.) are treated as characters.

Example 3

Operation: Change part of a line in a line numbered data set.

Known: The line number..... 143
The number of characters in the line preceding the characters to be changed..... 18

```
CHANGE 143 18
```

This form of the subcommand causes the first 18 characters of line number 143 to be listed at your terminal. You complete the line by typing the new information and enter the line by pressing the RETURN key. All of your changes will be incorporated into the data set.

Example 4

Operation: Change part of a particular line of a line numbered data set.

Known: The line number..... 103
A string of characters to be changed..... 315 h.p. at 2400

```
CHANGE 103 M315 h.p. at 2400
```

This form of the subcommand causes line number 103 to be searched until the characters "315 h.p. at 2400" are found. The line is displayed at your terminal up to the string of characters. You can then complete the line and enter the new version into the data set.

Example 5

Operation: Change the values in a table.

Known: The line number of the first line in the table..... 387
The line number of the last line in the table..... 406
The number of the column containing the values..... 53

```
CHANGE 387 406 52
```

Each line in the table is displayed at your terminal up to the column containing the value. As each line is displayed, you can type in the new value. The next line will not be displayed until you complete the current line and enter it into the data set.

Example 6

Operation: Add a sequence of characters to the front of the line that is currently referred to by the pointer within the system.

Known: The sequence of characters..... In the beginning

```
CHANGE * //In the beginning
```

Example 7

Operation: Delete a sequence of characters from a line-numbered data set.

Known: The line number containing the string of characters..... 15
The sequence of characters to be deleted..... weekly

```
CHANGE 15 /WEEKLY//
```

or

```
CHANGE 15 /WEEKLY/
```

Examples Using Quoted Strings

Example 1A

Operation: Change a sequence of characters in a particular line of a line numbered data set.

Known: The line number..... 57
The old sequence of characters..... parameter
The new sequence of characters..... operand

```
CHANGE 57 'parameter' 'operand'
```

Example 6A

Operation: Add a sequence of characters to the front of the line that is currently referred to by the pointer within the system.

Known: The sequence of characters..... In the beginning

```
CHANGE * '' 'In the beginning'
```

Example 7A

Operation: Delete a sequence of characters from a line-numbered data set.

Known: The line number containing the string of characters..... 15
The sequence of characters to be deleted..... weekly

```
CHANGE 15 'weekly' ''
```

Use the DELETE Subcommand to remove one or more records from the data set being edited.

Upon completion of the delete operation, the current-line pointer will point to the line that preceded the deleted line. If the first line of the data has been deleted, the current line pointer will be set to zero.

SUBCOMMAND	OPERANDS
{DELETE / D}	[* / line-number-1 [line-number-2] / *[count]]

line-number-1
specifies the line to be deleted; or the first line of a range of lines to be deleted.

line-number-2
specifies the last line of a range of lines to be deleted.

*
specifies that the first line to be deleted is the line indicated by the current line pointer in the system. This is the default if no line is specified.

count
specifies the number of lines to be deleted, starting at the location indicated by the preceding operand.

Example 1

Operation: Delete the line being referred to by the current-line pointer.

```
DELETE *
```

or

```
DELETE
```

or

```
D *
```

or

```
D
```

or

```
*
```

Any of the preceding command combinations or abbreviations will cause the deletion of the line referred to currently. The last instance is actually a use of the insert/replace/delete function, not the DELETE subcommand.

Example 2

Operation: Delete a particular line from the data set.

Known: The line number..... 04

```
DELETE 4
```

Leading zeroes are not required.

Example 3

Operation: Delete several consecutive lines from the data set.

Known: The number of the first line..... 18
The number of the last line..... 36

```
DELETE 18 36
```

Example 4

Operation: Delete several lines from a data set with no line numbers. The current line pointer in the system points to the first line to be deleted.

Known: The number of lines to be deleted..... 18

```
DELETE * 18
```

Example 5

Operation: Delete all the lines in a data set.

Known: The data set contains less than 100 lines and is not line-numbered.

```
TOP
DELETE * 100
```

Use the DOWN subcommand to change the current-line pointer so that it points to a line that is closer to the end of the data set.

SUBCOMMAND	OPERANDS
DOWN	[count]

count
specifies the number of lines toward the end of the data set that you want to move the current-line pointer. If you omit this operand, the default is one.

Example 1

Operation: Change the pointer so that it points to the next line.

```
DOWN
```

Example 2

Operation: Change the pointer so that you can refer to a line that is located closer to the end of the data set than the line currently pointed to.

Known: The number of lines from the present position to the new position..... 18

```
DOWN 18
```

END Subcommand of EDIT

Use the END subcommand to terminate operation of the EDIT command. After entering the END subcommand, you may enter new commands. If you have modified your data set and have not entered the SAVE subcommand, the system will ask you if you want to save the modified data set. If so, you can then enter the SAVE subcommand. If you do not want to save the data set, re-enter the END subcommand.

SUBCOMMAND	OPERANDS
END	

Use the FIND subcommand to locate a specified sequence of characters. The system begins the search at the line referred to by the current line pointer in the system, and continues until the character string is found or the end of the data set is reached.

SUBCOMMAND	OPERANDS
{FIND / F}	string [position]

string
specifies the sequence of characters (the character string) that you want to locate. This sequence of characters must be preceded by an extra character that serves as a special delimiter. The extra character may be any printable character other than a number, apostrophe, semicolon, blank, tab, comma, parenthesis, or asterisk. You must not use the extra character in the character string. Do not put a delimiter between the extra character and the string of characters.

Instead of using special delimiters to indicate a character string, you can use paired single quotes (apostrophes) to accomplish the same function with the FIND subcommand. The use of single quotes as delimiters for a character string is called quoted-string notation. Following are the rules for quoted-string notation for the string operand:

1. A string must be enclosed within single quotes, e.g., 'string character'.
2. A single quote within a character string is represented by two single quotes, e.g., 'pilgrims''s progress'.
3. A null string is represented by two single quotes, e.g., ''.

If you do not specify any operands, the operands you specified the last time you used FIND during this current usage of EDIT are used. The search for the specified string will begin at the line following the current line. Successive use of the FIND subcommand without operands allows you to search a data set, line by line.

position
specifies the column within each line at which you want the comparison for the string to be made. This operand specifies the starting column of the field to which the string is compared in each line.

If you want to consider a string starting in column 6, you must specify the digit 6 for the positional operand. When you use this operand with the special delimiter form of notation for "string", you must separate it from the string operand with the same special delimiter as the one preceding the string operand.

Example 1

Operation: Locate a sequence of characters in a data set.

Known: The sequence of characters..... ELSE GO TO COUNTER

```
FIND XELSE GO TO COUNTER
```

Example 2

Operation: Locate a particular instruction in a data set containing an assembler language program.

Known: The sequence of characters..... LA 3,BREAK
The instruction begins in column 10.

```
FIND 'LA    3,BREAK ' 10
```

Use the HELP subcommand to find out how to use EDIT subcommands. When you enter the HELP subcommand, the system responds by printing out explanatory information at your terminal. You may request:

- A list of available subcommands.
- An explanation of the function, syntax, and operands of a specific subcommand.

The HELP subcommand actually causes the system to execute a function of the HELP command; therefore, you may consult the discussion of the HELP command if you desire more detailed information.

SUBCOMMAND	OPERANDS
{HELP / H}	[subcommand-name [FUNCTION / SYNTAX / OPERANDS(list-of-operands) / ALL]]

subcommand-name
specifies the subcommand that you want to have clarified. If you omit this operand, the system will display a list of EDIT subcommands.

FUNCTION
specifies that you want a description of the referenced subcommand's function.

SYNTAX
specifies that you want a definition of the proper syntax for the referenced subcommand.

OPERANDS(list-of-operands)
specifies that you want an explanation of the operands applicable to the referenced subcommand. The list of operands specifies the particular keywords that you want to have explained. If you do not specify any keywords, all keywords and positional operands will be included.

ALL
specifies that you want a description of the function, the syntax, and the operands of the subcommand that you specified. This is the default if no operand is specified with the subcommand name.

Example 1

Operation: Have a list of available subcommands displayed at your terminal.

```
HELP
```

Example 2

Operation: Obtain all available information about a particular subcommand.

Known: The subcommand name..... FIND

```
H FIND
```

Example 3

Operation: Have a description of each operand for a particular subcommand displayed at your terminal.

Known: The subcommand name..... LIST

```
h list operands
```

Use the INPUT subcommand to put the system in input mode so that you can add or replace data in the data set being edited.

SUBCOMMAND	OPERANDS
{INPUT / I}	[line-number [increment] / *] [R / I] [PROMPT / NOPROMPT]

line-number
specifies the line number and location for the first new line of input. If no operands are specified, input data will be added to the end of the data set.

increment
specifies the amount that you want each succeeding line number to be increased. If you omit this operand, the default is the last increment specified with the INPUT or RENUM subcommand. If neither of these subcommands has been specified with an increment operand, an increment of 10 will be used.

*
specifies that the next new line of input will either replace or follow the line pointed to by the current-line pointer, depending on whether you specify the R or I operand. If an increment is specified with this operand, it is ignored.

R
specifies that you want to replace existing lines of data and insert new lines into the data set. This operand is ignored if you fail to specify either a line number or an asterisk. If the specified line already exists, the old line will be replaced by the new line. If the specified line is vacant, the new line will be inserted at that location.

I
specifies that you want to insert new lines into the data set without altering existing lines of data. This operand is ignored if you fail to specify either a line number or an asterisk.

PROMPT
specifies that you want the system to display either a line number or, if the data set is not line-numbered, a prompting character before each new input line. If you omit this operand, the default is:

a. The value (either PROMPT or NOPROMPT) that was established the last time you used input mode.
b. PROMPT, if this is the first use of input mode and the data set has line numbers.
c. NOPROMPT, if this is the first use of input mode and the data set does not have line numbers.

NOPROMPT
specifies that you do not want to be prompted.

Example 1

Operation: Add and replace data in an old data set.

Known: The data set is to contain line numbers.
Prompting is desired.
The ability to replace lines is desired.
The first line number..... 2
The increment value for line numbers..... 2

```
INPUT 2 2 R PROMPT
```

The listing at your terminal will resemble the following sample listing with your input in lower case and the computers response in upper case.

```
edit query cobol old

EDIT

input 2 2 r prompt

INPUT

00002 identification division
00004 program-id.query
00006
```

Example 2

Operation: Insert data in an existing data set.

Known: The data set contains text for a report.
The data set does not have line numbers.
The ability to replace lines is not necessary.
The first input data is "New research and development activities will" which is to be placed at the end of the data set.

```
INPUT
```

The listing at your terminal will resemble the following sample listing:

```
edit forecast.text old text nonum asis
EDIT
input
INPUT
New research and development activities will
```

Use the INSERT subcommand to insert one or more new lines of data into the data set. Input data is inserted following the location pointed to by the line pointer in the system. (If no operands are specified, input data will be placed in the data set line following the current line.) You may change the position pointed to by the line pointer by using the BOTTOM, DOWN, TOP, UP, FIND and LIST subcommands.

SUBCOMMANDS	OPERANDS
{INSERT} {IN }	[insert-data]

insert-data
specifies the complete sequence of characters that you wish to insert into the data set at the location indicated by the line pointer. When the first character of the inserted data is a tab, no delimiter is required between the command and the data. Only a single delimiter is recognized by the system. If you enter more than one delimiter, all except the first are considered to be input data.

Example 1

Operation: Insert a single line into a data set.

Known: The line to be inserted is:

"UCBFLG DS AL1 CONTROL FLAGS"

The location for the insertion follows the 71st line in the data set.

The current line pointer points to the 74th line in the data set.

The user is operating in EDIT mode.

Before entering the INSERT subcommand, the current line pointer must be moved up 3 lines to the location where the new data will be inserted.

```
UP 3
```

The INSERT subcommand is now entered.

```
INSERT UCBFLG DS AL1 CONTROL FLAGS
```

The listing at your terminal will be similar to the following sample listing.

```
up 3
insert ucbflg ds al1 control flags
```

Example 2

Operation: Insert several lines into a data set.

Known: The data set contains line numbers.
The inserted lines are to follow line number 00280.
The current line pointer points to line number 00040.
The user is operating in EDIT mode.
The lines to be inserted are:
"J.W.HOUSE 13-244831 24.73"
"T.N.HOWARD 24-782095 3.05"
"B.H.IRELAND 04-007830 104.56"

Before entering the INSERT subcommand the current line pointer must be moved down 24 lines to the location where the new data will be inserted.

```
DOWN 24
```

The INSERT subcommand is now entered.

```
INSERT
```

The system will respond with

```
INPUT
```

The lines to be inserted are now entered.

```
J.W. House   13-244831  24.73
T.N. Howard  24-782095   3.05
B.H. Ireland 04-007830 104.56
```

The listing at your terminal will be similar to the following sample listing:

```
down 24
insert
INPUT
00281 j.w.house   13-244831 24.73
00282 t.n.howard  24-782095 3.05
00283 b.h.ireland 04-007830 104.56
```

New line numbers are generated in sequence beginning with the number one greater than the one pointed to by the current line pointer. When no line can be inserted, you will be notified. No resequencing will be done.

The INSERT/REPLACE/DELETE function lets you insert, replace, or delete a line of data without specifying a subcommand name. To insert or replace a line, all you need to do is indicate the location and the new data. To delete a line, all you need to do is indicate the location. You can indicate the location by specifying a line number or an asterisk. The asterisk means that the location to be used is pointed to by the line pointer within the system. You can change the line pointer by using the UP, DOWN, TOP, BOTTOM, and FIND subcommands so that the proper line is referred to.

SUBCOMMAND	OPERANDS
	{line-number | *} [string]

line-number
: specifies the number of the line you want to insert, replace, or delete.

*
: specifies that you want to replace or delete the line at the location pointed to by the line pointer in the system. You can use the TOP, BOTTOM, UP, DOWN, and FIND subcommands to change the line pointer without modifying the data set you are editing.

string
: specifies the sequence of characters that you want to either insert into the data set or to replace an existing line. If this operand is omitted and a line exists at the specified location, the existing line is deleted. When the first character of "string" is a tab, no delimiter is required between this operand and the preceding operand. Only a single delimiter is recognized by the system. If you enter more than one delimiter, all except the first are considered to be input data.

How the System Interprets the Operands: When you specify only a line number or an asterisk, the system deletes a line of data. When you specify a line number or asterisk followed by a sequence of characters, the system will replace the existing line with the specified sequence of characters or, if there is no existing data at the location, the system will insert the sequence of characters into the data set at the specified location.

Example 1

Operation: Insert a line into a data set.

Known: The number to be assigned to the new line..... 62
The data..... "OPEN INPUT PARTS-FILE"

```
|     62  OPEN INPUT PARTS-FILE                                          |
```

Example 2

Operation: Replace an existing line in a data set.

Known: The number of the line that is to be replaced..... 287
The replacement data..... "GO TO HOURCOUNT;"

```
|     287  GO TO HOURCOUNT;                                              |
```

Example 3

Operation: Replace an existing line in a data set that does not have line numbers.

Known: The line pointer in the system points to the line that is to be replaced.
The replacement data is..... "58 PRINT USING 120,S"

```
|     *  58 PRINT USING 120,S                                            |
```

Example 4

Operation: Delete an entire line.

Known: The number of the line..... 98
The current line pointer in the system points to line 98.

```
|98                                                                      |
```

or

```
|*                                                                       |
```

Use the LIST subcommand to display one or more lines of your data set at your terminal.

SUBCOMMAND	OPERANDS
{LIST / L}	[line-number-1 [line-number-2] / * [count]]
	[NUM / SNUM]

line-number-1
specifies the number of the line that you want to be displayed at your terminal.

line-number-2
specifies the number of the last line that you want displayed. When you specify this operand, all the lines from line number 1 through line number 2 are displayed.

*
specifies that the line referred to by the line pointer in the system is to be displayed at your terminal. You can change the line pointer by using the UP, DOWN, TOP, BOTTOM, and FIND subcommands without modifying the data set you are editing.

count
specifies the number of lines that you want to have displayed, starting at the location referred to by the line pointer.
Note: If you do not specify any operand with LIST, the entire data set will be displayed.

NUM
specifies that line numbers are to be displayed with the text. This is the default value if both NUM and SNUM are omitted. If your data set does not have line numbers, this operand will be ignored by the system.

SNUM
specifies that line numbers are to be suppressed, i.e., not printed on the listing.

Example 1

Operation: List an entire data set.

```
LIST
```

Example 2

Operation: List part of a data set that has line numbers.

Known: The line number of the first line to be displayed..... 27
The line number of the last line to be displayed..... 44
Line numbers are to be included in the list.

```
LIST 27 44
```

Example 3

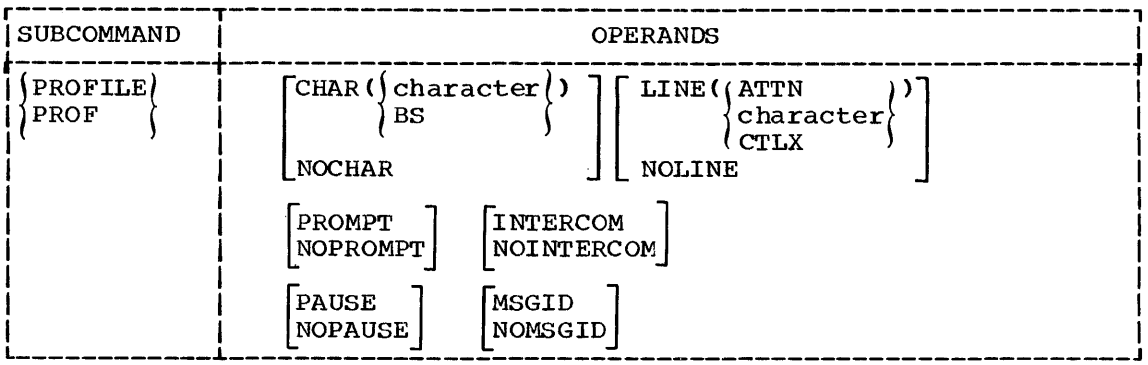
Operation: List part of a data set that does not have line numbers.

Known: The line pointer in the system points to the first line to be listed.
The section to be listed consists of 17 lines.

```
LIST * 17
```

Use the PROFILE subcommand to specify the character-deletion and/or line-deletion indicators that you want to use at your terminal.

Initially, a user profile is prepared for you when arrangements are made for you to use the system. You change the characteristics of your user profile by using the PROFILE subcommand with the appropriate operands. Only the characteristics that you specify by operands will change; other characteristics remain unchanged. You must specify at least one operand or the subcommand will be ignored by the system. (See the PROFILE command.)



CHAR(BS or character)
specifies the character or terminal keyboard key that you want to use at your terminal to delete a character from a line.

BS specifies that the backspace key is to be your character-deletion indicator. (This is the initial value that is in effect until changed specifically.)

Character specifies the particular character that you want to use as your character-deletion indicator. You should not specify a blank, comma, tab, asterisk, parenthesis, colon, apostrophe or any of the characters in the subcommand name (profile) or in any of its operands.

Note: Do not use an alphabetic character as either a character delete, or a line delete, character. If you do, you run the risk of not being able to enter certain commands without accidentally deleting characters or lines of data. For instance; if you specify R as a character delete character, each time you tried to enter a PROFILE subcommand the R in PROFILE would delete the P that precedes it. Thus it would be impossible to enter the PROFILE subcommand as long as R was the character delete control character.

NOCHAR
specifies that you do not want to use the character-deleting indicator.

LINE(ATTN, character, or CTLX)
specifies the character or key that you want to use at your terminal to delete an entire line. You should not specify a blank, comma, tab, asterisk, parenthesis, colon, or apostrophe.

ATTN specifies that an attention interruption is to delete a line. (This is the initial value that is in effect until changed specifically.)

Character specifies the particular character or key that you want to use as your line deletion indicator.

CTLX specifies that for a teletype terminal the X and CTRL keys are to be interpreted as a line-deletion character.

NOLINE
specifies that you do not want to use the line-deletion indicator.

Example 1

Operation: Specify that the backspace key is used for deleting a character and that the ATTN key is used for deleting a line.

```
PROFILE  CHAR(BS) LINE(ATTN)
```

Example 2

Operation: Specify that an exclamation mark is used for deleting a character and that a pound sign is used for deleting a line.

```
PR  CHAR(!) LINE(#)
```

Use the RENUM subcommand to:

- Assign a line number to each record of a data set that does not have line numbers.
- Renumber each record in a data set that has line numbers.

New line numbers are placed in the last eight character positions of fixed length records (except for COBOL), or in the first eight character positions of variable length records. Line numbers for COBOL data sets are placed in the first six positions. The default line number position for ASM data sets is from column 73 through 80. However, by specifying the NUM operand, you can position the line number anywhere within this field. If variable length records were not numbered previously, the records will be lengthened so that the eight-character fields can be prefixed to each record. If the record cannot be extended eight characters, you are notified. Any information in the last positions of fixed length records (or the first 6 positions of COBOL data records) is replaced by the line numbers.

In all cases the specified (or default) increment value becomes the line increment for the data set.

SUBCOMMAND	OPERANDS
{ RENUM / REN }	[new-line-number [increment [old-line-number]]]

new line number
: specifies the first line number to be assigned to the data set. If this operand is omitted, the first line number will be 10.

increment
: specifies the amount by which each succeeding line number is to be incremented. (The default value is 10.) You cannot use this operand unless you specify a new line number.

old-line-number
: specifies the location within the data set where renumbering will begin. If this operand is omitted, renumbering will start at the beginning of the data set. You cannot use this operand unless you specify a value for the increment operand or when you are initially numbering a NONUM data set.

Example 1

Operation: Renumber an entire data set.

```
RENUM
```

Example 2

Operation: Renumber part of a data set.

Known: The old line number..... 17
The new line number..... 21
The increment..... 1

```
REN 21 1 17
```

Example 3

Operation: Renumber part of a data set from which lines have been deleted.

Known: Before deletion of the lines, the data set contained lines 10, 20, 30, 40, and 50.
Lines 20 and 30 were deleted.
Lines 40 and 50 are to be renumbered with an increment of 10.

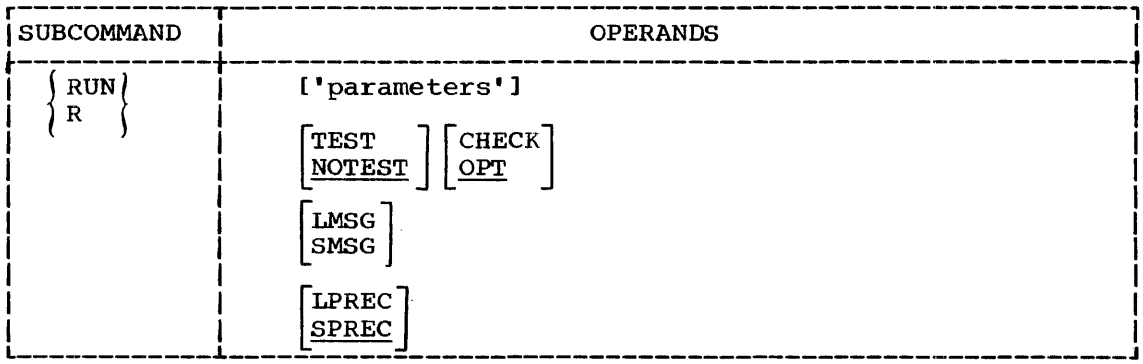
```
REN 20 10 40
```

Note: The lowest acceptable value for a new line number in this example is 11.

Use the RUN subcommand to compile, load, and execute the source statements in the data set that you are editing. The RUN subcommand is designed specifically for use with certain program products; it selects and invokes the particular program product needed to process your source statements. The following table shows which program product is selected to process each type of source statement. (Appendix A contains references to additional information about the program products.)

Note: Any data sets required by your problem program should be allocated before you enter EDIT mode.

If your program or data set contains statements of this type (see EDIT):	Then the following Program Product is needed:
ASM	TSO ASM Prompter
BASIC	ITF:BASIC (Shared Language Component and BASIC Processor)
COBOL	TSO COBOL Prompter and American National Standard COBOL Version 3
FORTGI	TSO FORTRAN Prompter and FORTRAN IV(G1)
GOFORT	Code and Go FORTRAN
IPLI	ITF:PL/I (Shared Language Component and PL/I Processor)
PLI	PL/I Checkout Compiler or PL/I Optimizing Compiler.
Programs containing statements suitable for the following IBM-supplied language processors can be compiled and executed by using the CALL command. ASM(F), PL/1(F), FORTRAN(E), (G) or (H) You can use the CONVERT command to convert ITF:PL/I and Code and Go FORTRAN statements to a form suitable for the PL/1 and FORTRAN compilers, respectively. When the descriptive qualifier for your data set name is .FORT, the Code and Go Fortran compiler is invoked unless you specify another compiler with the EDIT command. Note: User-defined data set types can be executed under the RUN subcommand of EDIT if a prompter name was specified at system generation time. The RUN command will not recognize these same data sets.	



'parameters'
specifies a string of up to 100 characters that is passed to the program that is to be executed. You may specify this operand only for programs which can accept parameters. Observe the standard Operating System conventions as described in IBM System/360 Operating System Supervisor Services and Macro Instructions, GC28-6646.

TEST
specifies that testing will be performed during execution. This operand is valid for ITF:PL/I and ITF:BASIC Program Product programs only.

NOTEST
specifies that no testing will be done. If you omit both TEST and NOTEST, the default value is NOTEST.

LMSG
specifies that you want to receive complete diagnostic messages. This operand is valid for the optional ITF:PL/I, ITF:BASIC and Code and Go FORTRAN Program Products only.

Note: The default value for the LMSG/SMSG operand pair depends on the Program Product being used, as follows:

Program Product	Default Operand
Code and Go	SMSG
ITF:BASIC	LMSG
ITF:PL/I	LMSG

SMSG
specifies that you want to receive the short, concise diagnostic messages.

LPREC
specifies that you want long precision arithmetic calculations (valid only for the ITF:BASIC Program Product).

SPREC
specifies that you want short precision arithmetic calculations (valid only for the ITF:BASIC Program Product). If you omit both LPREC and SPREC, the default value is SPREC.

CHECK
specifies the PL/I Checkout Compiler. This operand is valid for the PL/I Program Product only. If you omit this operand, the OPT operand is the default value.

OPT
specifies the PL/I Optimizing Compiler. This operand is valid for the PL/I Program Product only. This is the default value if both CHECK and OPT are omitted.

Example 1

Operation: Compile and execute the data being edited by the EDIT command.

Known: The EDIT command is being used currently.
The data set contains statements prepared for the optional ITF:BASIC Program Product compiler.
The system contains the optional ITF:BASIC Program Product.
Default values for the RUN subcommand are suitable.

```
RUN
```

Example 2

Operation: Execute an assembler language program contained in the data set referred to by the EDIT command.

Known: The parameters to be passed to the program are: '1024,PAYROLL'

```
RUN '1024,PAYROLL'
```

SAVE Subcommand of EDIT

Use the SAVE subcommand to have your data set retained as a permanent data set. If you use SAVE without an operand, the updated version of your data set replaces the original version. When you specify a new data set name as an operand, both the original version and the updated version of the data set are available for further use.

SUBCOMMAND	OPERANDS
{SAVE / S}	[data-set-name]

data-set-name
specifies a data set name that will be assigned to your edited data set. The new name may be different from the current name. (See the data set naming conventions.) If this operand is omitted, the name entered with the EDIT command will be used.

If you specify the name of an existing data set or a member of a partitioned data set, that data set or member is replaced by the edited data set. (Before replacement occurs, you will be given the option of specifying a new data set name or member name.)

If you do not specify the name of an existing data set or partitioned data set member, a new data set (the edited data set) will be created with the name you specified. If you specified a member name for a sequentially-organized data set, no replacement of the data set will take place. If you do not specify a member name for an existing partitioned data set, the edited data set is assigned a member name of TEMPNAME.

Example 1

Operation: Save the data set that has just been edited by the EDIT command.

Known: The system is in edit mode.
The user supplied name that you want to give the data set is INDEX.

```
SAVE  INDEX
```

Use the SCAN subcommand to request syntax checking services for statements that will be processed by the PL/I(F), FORTRAN(E), FORTRAN(G), or FORTRAN(H) compiler or by the Code and Go FORTRAN, FORTRAN IV (GI), PL/I Checkout and Optimizing compiler, ITF:BASIC or ITF:PL/I Program Products. You can have each statement checked as you enter it in Input mode, or any or all existing statements checked. You cannot check the syntax of statements that you are adding, replacing, or modifying, via the CHANGE subcommand, the INSERT subcommand with the insert-data operand, or the insert/replace/delete function unless the statements are written in ITF:BASIC or ITF:PLI.

SUBCOMMAND	OPERANDS
{SCAN / SC}	[line-number-1 [line-number-2]] / [* [count]]
	[ON / OFF]

line-number-1
: specifies the number of a line to be checked for proper syntax.

line-number-2
: specifies that all lines between line number 1 and line number 2 are to be checked for proper syntax.

*
: specifies that the line at the location indicated by the line pointer in the system is to be checked for proper syntax. The line pointer can be changed by the TOP, BOTTOM, UP, DOWN, and FIND subcommands.

count
: specifies the number of lines, beginning with the current line, that you want checked for proper syntax.

ON
: specifies that each line is to be checked for proper syntax as it is entered in Input mode.

OFF
: specifies that each line is not to be checked as it is entered in Input mode.

NOTE: If no operands are specified, all existing statements will be checked for proper syntax.

Example 1

Operation: Have each line of a FORTRAN program checked for proper syntax as it is entered.

```
SCAN  ON
```

Example 2

Operation: Have all the statements in a data set checked for proper syntax.

```
SCAN
```

Example 3

Operation: Have several statements checked for proper syntax.

Known: The number of the first line to be checked..... 62
The number of the last line to be checked..... 69

```
SCAN  62 69
```

Example 4

Operation: Check several statements for proper syntax.

Known: The line pointer points to the first line to be checked.
The number of lines to be checked..... 7

```
SCAN  *  7
```

Use the TABSET subcommand to:

- Establish or change the logical tabulation settings.
- Void any existing tabulation settings.

The basic form of the subcommand causes each strike of the tab key to be translated into blanks corresponding to the column requirements for the data set type. For instance, if the name of the data set being edited has FORT as a descriptive qualifier, the first tabulation setting will be in column 7. The values in Figure 11 will be in effect when you first enter the EDIT command.

Data Set Name Descriptive Qualifier	Default Tab Settings Columns
ASM	10,16,31,72
BASIC(ITF:BASIC Program Product)	10,20,30,40,50,60
CLIST	10,20,30,40,50,60
CNTL	10,20,30,40,50,60
COBOL	8,12,72
DATA	10,20,30,40,50,60
FORT (FORTRAN(E), FORTRAN(G), FORTRAN(H), compilers, FORTRAN IV (GI) and Code and Go Fortran Program Product data set types.)	7,72
IPLI(ITF:PL/I Program Product)	5,10,15,20,25,30,35,40,45,50
PLI (PLI(F), and PLI checkout and optimizing compiler data set types).	5,10,15,20,25,30,35,40,45,50
TEXT	5,10,15,20,30,40
User-defined	10,20,30,40,50,60

Figure 11. Default Tab Settings

You may find it convenient to have the mechanical tab settings coincide with the logical tab settings. This can be accomplished by realizing that, except for line-numbered COBOL data sets, the logical tab columns apply only to the data that you actually enter. Since a printed line number prompt is not logically part of the data you are entering, the logical tab positions are calculated beginning at the next position after the prompt. Thus, if you are receiving five-digit line number prompts and have set a logical tab in column 10, the mechanical tab should be set 15 columns to the right of the margin. If you are not receiving line number prompts, the mechanical tab should be set 10 columns to the right of the margin.

In COBOL data sets the sequence number (line number) is considered to be a logical (as well as physical) part of each record that you enter. For instance, if you specify the NONUM operand on the EDIT command, while editing a COBOL data set, the system assumes that column 1 is at the left margin and that you are entering the required sequence numbers in the first six columns; thus, logical tabs are calculated from the left margin (column 1). In line-numbered COBOL data sets (the NONUM operand was not specified), the column following a line number prompt is considered to be column 7 of your data - the first 6 columns being occupied by the system-supplied sequence number(line number).

SUBCOMMAND	OPERANDS
{TABSET / TAB}	[ON [(integer-list)] / OFF / IMAGE]

ON(integer-list)
specifies that tab settings are to be translated into blanks by the system. If you specify ON without an integer list, the existing or default tab settings are used. You can establish new values for tab settings by specifying the numbers of the tab columns as values for the integer list. A maximum of ten values is allowed. If you omit both ON and OFF the default value is ON.

OFF
specifies that there is to be no translation of tabulation characters. Each strike of the tab key will produce a single blank in the data.

IMAGE
specifies that the next input line will define new tabulation settings. The next line that you type should consist of "t"s, indicating the column positions of the tab settings, and blanks or any other characters except "t". 10 settings is the maximum number of tabs allowable. Do not use the tab key to produce the new image line. A good practice is to use a sequence of digits between the "t"s so you can easily determine which columns the tabs are set to. (See example 3.)

Example 1

Operation: Re-establish standard tab settings for your data set.

Known: Tab settings are not in effect.

```
TAB
```

Example 2

Operation: Establish tabs for columns 2, 18, and 72.

```
TAB  ON(2 18 72)
```

Example 3

Operation: Establish tabs at every 10th column.

```
TAB IMAGE
123456789t12345789t123...
```

Use the TOP subcommand to change the line pointer in the system to zero. That is, the pointer will point to the position preceding the first line of an unnumbered data set or of a numbered data set which does not have a line number of zero. The pointer will point to line number zero of a data set that has one.

This subcommand is useful in setting the line pointer to the proper position for subsequent subcommands that need to start their operations at the beginning of the data set.

In the event that the data set is empty you will be notified but the current line pointer still takes on a zero value.

SUBCOMMAND	OPERANDS
TOP	

Example 1

Operation: Move the line pointer to the beginning of your data set.

Known: The data set is not line-numbered.

```
top
LINE NUMBER '0' NOT FOUND
```

Use the UP subcommand to change the line pointer in the system so that it points to a record nearer the beginning of your data set. If the use of this subcommand causes the line pointer to point to the first record of your data set, you will be notified.

SUBCOMMAND	OPERANDS
UP	[count]

count
specifies the number of lines toward the beginning of the data set that you want to move the current line pointer. If count is omitted, the pointer will be moved only one line.

Example 1

Operation: Change the pointer so that it refers to the preceding line.

```
UP
```

Example 2

Operation: Change the pointer so that it refers to a line located 17 lines before the location currently referred to.

```
UP 17
```

Use the VERIFY subcommand to display the line that is currently pointed to by the line pointer in the system; whenever the current line pointer has been moved, or whenever a line has been modified by use of the CHANGE subcommand. Until you enter VERIFY, you will have no verification of changes in the position of the current line pointer.

SUBCOMMAND	OPERANDS
{VERIFY / V}	[ON / OFF]

ON
specifies that you want to have the line that is referred to by the line pointer displayed at your terminal each time the line pointer changes or each time the line is changed by the CHANGE subcommand. This is the default if you omit both ON and OFF.

OFF
specifies that you want to discontinue this service.

Example 1

Operation: Have the line that is referred to by the line pointer displayed at your terminal each time the line pointer changes.

```
VERIFY
```

or

```
VERIFY ON
```

Example 2

Operation: Terminate the operations of the VERIFY subcommand.

```
VERIFY OFF
```

Use the EXEC command to execute a command procedure (see section entitled "Command Procedure Statements").

You can specify the EXEC command in two ways:

1. The explicit form, where you enter EXEC followed by the name of the data set that contains the command procedure.

2. The implicit form, where you do not enter EXEC but only enter the name of the member of the command procedure library (a partitioned data set) that contains the command procedure.

Some of the commands in a command procedure may have symbolic values for operands. When you specify the EXEC command, you may supply actual values for the system to use in place of the symbolic values.

COMMAND	OPERANDS
{EXEC / EX}	data-set-name ['value-list'] [NOLIST / LIST]
	procedure-name [value-list]

data-set-name
specifies the name of the data set containing the command procedure to be executed. If the descriptive qualifier for the data set is not CLIST (as in BOB.FORTCOMP.CLIST) you must enclose the fully qualified name within apostrophes. (See the data set naming conventions.)

procedure-name
specifies a member of a command procedure library that is invoked when you enter the LOGON command. The library must previously have been defined in the SYSPROC DD statement of the logon procedure or with the ALLOCATE command.

value-list
specifies the actual values that are to be substituted for the symbolic values in the command procedure. The symbolic values are defined by the operands of the PROC statement in the command procedure. The actual values that are to replace the symbolic values defined by positional operands in the PROC statement must be in the same sequence as the positional operands. The actual values that are to replace the symbolic values defined by keywords in the PROC statement must follow the positional values, but may be in any sequence. When you use the explicit form of the command, the value list must be enclosed in apostrophes. If apostrophes appear within the list, then you must provide two apostrophes in order to print one.

NOLIST
specifies that the commands and subcommands will not be listed at the terminal. The system assumes NOLIST for implicit and explicit EXEC commands.

LIST
specifies that commands and subcommands will be listed at the terminal as they are executed. This operand is valid only for the explicit form of EXEC.

Example 1

Operation: Execute a command procedure to invoke the PL/I compiler.

Known: The name of the data set that contains the command procedure is RBJ2I.PLIR.CLIST.

The command procedure consists of:

```
PROC 1 NAME
ALLOCATE DATASET(&NAME..PLI) FILE(SYSIN)
ALLOCATE DATASET(&NAME..LIST) FILE(SYSPRINT) BLOCK(80) SPACE(300,100)
ALLOCATE DATASET(&NAME..OBJ) FILE(SYSLIN) BLOCK(80) SPACE(250,100)
ALLOCATE FILE(SYSUT1) BLOCK(1024) SPACE(60,60)
ALLOCATE FILE(SYSUT3) BLOCK(80) SPACE(250,100)
CALL 'SYS1.LINKLIB(IEMAA)' 'LIST,ATR,XREF,STMT'
FREE FILE(SYSUT1,SYSUT3,SYSIN,SYSPRINT)
```

The name of your program is 'EXP'.
You want to have the names of the commands in the command procedure displayed at your terminal as they are executed.

```
EXEC PLIR 'EXP' LIST
```

The listing at your terminal will be similar to:

```
exec plir 'exp' list

ALLOCATE DATASET(EXP.PLI) FILE(SYSIN)
ALLOCATE DATASET(EXP.LIST) FILE(SYSPRINT) BLOCK(80) SPACE(300,100)
ALLOCATE DATASET(EXP.OBJ) FILE(SYSLIN) BLOCK(80) SPACE(250,100)
ALLOCATE FILE(SYSUT1) BLOCK(1024) SPACE(60,60)
ALLOCATE FILE(SYSUT3) BLOCK(80) SPACE(250,100)
CALL 'SYS1.LINKLIB(IEMAA)' 'LIST,ATR,XREF,STMT'
FREE FILE(SYSUT1,SYSUT3,SYSIN,SYSPRINT)
READY
```

Example 2

Operation: Suppose that the command procedure in Example 1 was stored in a command procedure library. Execute the command procedure using the implicit form of EXEC.

Known: The name of the member of the partitioned data set that contains the command procedure is PLIR

```
plir exp
```

Use the FREE command to release ("de-allocate") previously allocated data sets that you no longer need. You can also use this command to change the output class of SYSOUT data sets and to delete attribute lists.

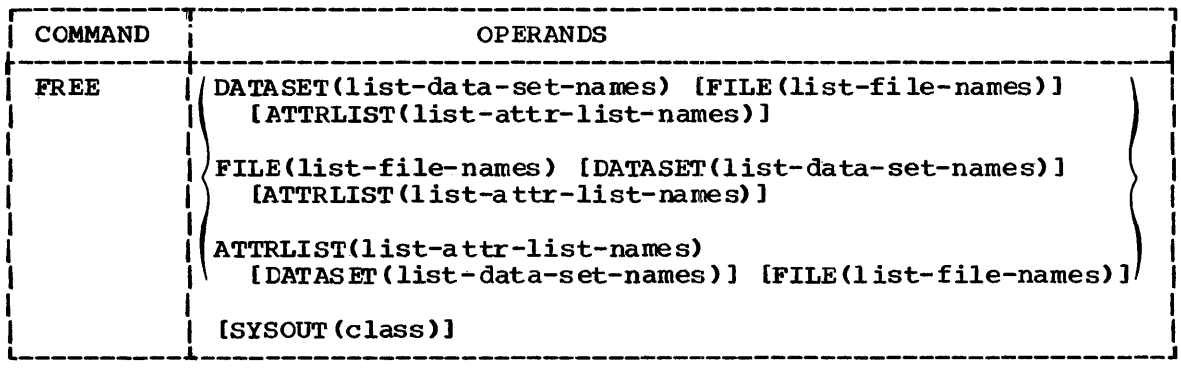
The maximum number of data sets that may be allocated to you at any one time depends on the number of Data Definition (DD) statements in the procedure that is invoked when you LOGON. The allowable number must be large enough to accomodate:

- Data sets allocated via the LOGON and ALLOCATE commands.
- Data sets allocated dynamically, and later freed automatically, by the system's command processors.

The data sets allocated by the LOGON and ALLOCATE commands are not freed automatically. To avoid the possibility of reaching your limit and being denied necessary resources, you should use the FREE command to release these data sets when they are no longer needed.

When you free SYSOUT data sets, you may change their output class to make them available for processing by an output writer.

When you enter the LOGOFF command, all data sets allocated to you and attribute lists created during the terminal session are freed by the system.



COMMAND	OPERANDS
FREE	DATASET(list-data-set-names) [FILE(list-file-names)] [ATTRLIST(list-attr-list-names)] FILE(list-file-names) [DATASET(list-data-set-names)] [ATTRLIST(list-attr-list-names)] ATTRLIST(list-attr-list-names) [DATASET(list-data-set-names)] [FILE(list-file-names)] [SYSOUT(class)]

DATASET(list-of-data-set-names)
: specifies one or more data set names that identify the data sets that you want to free. The data set name must include the descriptive (rightmost) qualifier and may contain a member name in parentheses. (See the data set naming conventions.) If you omit this operand, you must specify either the FILE or the ATTRLIST operand.

FILE(list-of-file-names)
: specifies one or more file names that identify the data sets to be freed. If you omit this operand, you must specify either the DATASET or the ATTRLIST operand.

ATTRLIST(list-of-attrlist-names)
: specifies the names of one or more attribute lists that you want to delete. If you omit this operand, you must specify either the DATASET or the FILE operand.

SYSOUT(class)
specifies an output class which is represented by a single character. All of the system output (SYSOUT) data sets specified in the DATASET and FILE operands will be assigned to this class and placed in the output queue for processing by an output writer (see IBM System/360 Operating System: Supervisor Services and Macro Instructions, GC28-6646, and Data Management Services, GC26-3746. In order to free a file to SYSOUT, the file must have previously been allocated to SYSOUT.

Example 1

Operation: Free a data set by specifying its data set name.

Known: The data set name..... T0C903.PROGA.LOAD

```
FREE DATASET(PROGA.LOAD)
```

Example 2

Operation: Free three data sets by specifying their data set names.

Known: The data set names..... LIRPA.PB99CY.ASM
LIRPA.FIRSTQTR.DATA
LIRPA.LOOF.MSG

```
FREE DATASET(PB99CY.ASM,FIRSTQTR.DATA,'LIRPA.LCOF.MSG')
```

Example 3

Operation: Free five data sets by specifying data set names or data definition names. Change the output class for any SYSOUT data sets being freed.

Known: The name of a data set..... DNIW.HCRAM.FORT
The filenames (data definition names) of
4 data sets..... SYSUT1
SYSUT3
SYSIN
SYSPRINT
The new output class..... B

```
FREE DATASET(HCRAM.FORT) FILE(SYSUT1,SYSUT3,SYSIN,SYSPRINT) SYSOUT(B)
```

Example 4

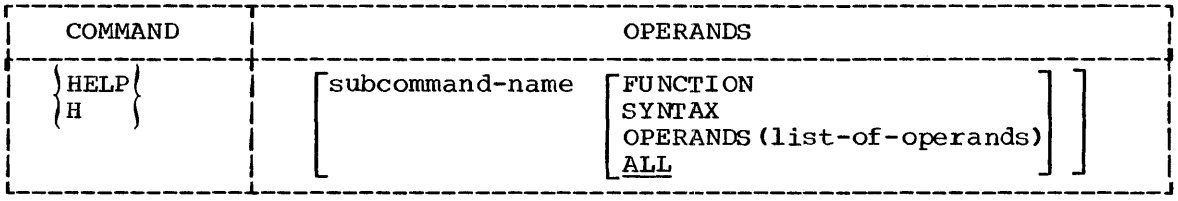
Operation: Delete two attribute lists.

Known: The names of the lists..... DCBPARMS
ATTRIBUT

```
FREE ATTRLIST(DCBPARMS ATTRIBUT)
```

HELP Command

Use the HELP command to obtain information about the function, syntax, and operands of commands and subcommands. This reference information is contained within the system and is displayed at your terminal in response to your request for help.



command-name
specifies the name of the command that you want to know more about.

FUNCTION
specifies that you want to know more about the purpose and operation of the command.

SYNTAX
specifies that you want to know more about the syntax required to use the command properly.

OPERANDS(list-of-operands)
specifies that you want to see explanations of the operands for the command. When you specify the keyword OPERANDS and omit any values, all operands will be described. You can specify particular keyword operands that you want to have described by including them as values within parenthesis following the keyword. If you specify a list of more than one operand, the operands in the list must be separated by commas or blanks.

ALL
specifies that you want to see all information available concerning the command or subcommand. This is the default value if no other KEYWORD operand is specified.

HELP Information: The scope of available information ranges from general to specific. The HELP command with no operands produces a list of valid commands and their basic functions. From the list you can select the command most applicable to your needs. If you need more information about the selected command, you may use the HELP command again, specifying the selected command name as an operand. You will then receive:

1. A brief description of the function of the command.
2. The format and syntax for the command.
3. A description of each operand.

You can obtain information about a command only when the system is ready to accept a command.

If you do not want to have all of the detailed information, you may request only the portion that you need.

The information about the commands is contained in a cataloged partitioned data set named SYS1.HELP. Information for each command is kept in a member of the partitioned data set. The HELP command causes the system to select the appropriate member and display its contents at your terminal.

Figure 12 shows the hierarchy of the sets of information available with the HELP command. Figure 12 also shows the form of the command necessary to produce any particular set.

Example 1

Operation: Obtain a list of all available commands.

```
HELP
```

Example 2

Operation: Obtain all the information available for the ALLOCATE command.

```
HELP ALLOCATE
```

Example 3

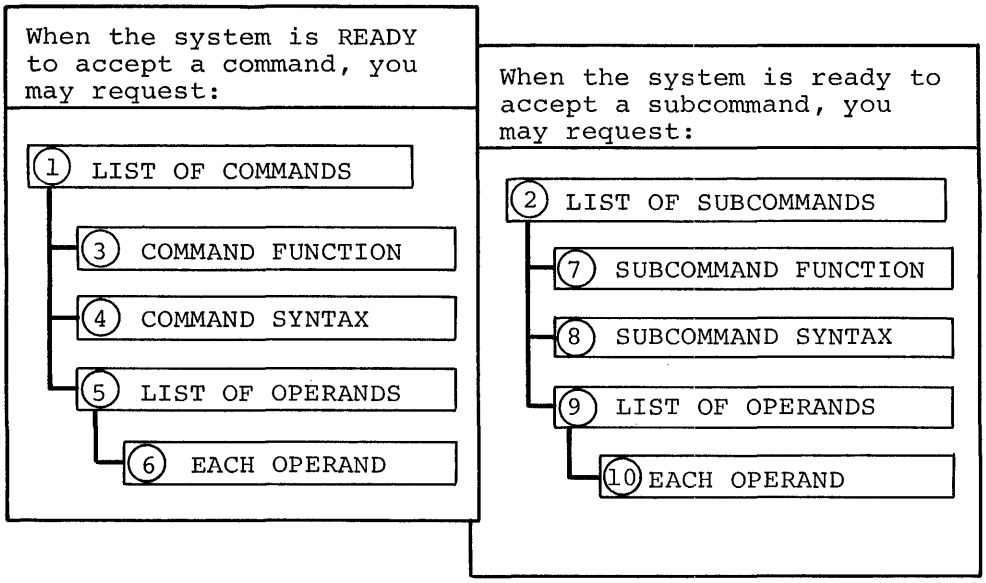
Operation: Have a description of the XREF, MAP, COBLIB, and OVLY operands for the LINK command displayed at your terminal.

```
H LINK OPERANDS(XREF,MAP,COBLIB,OVLY)
```

Example 4

Operation: Have a description of the function and syntax of the LISTBC command displayed at your terminal.

```
h listbc function syntax
```



this form of the command	produces:
READY mode	
HELP	1
HELP commandname	3 4 5
HELP commandname ALL	3 4 5
HELP commandname FUNCTION	3
HELP commandname SYNTAX	4
HELP commandname OPERANDS	5
HELP commandname OPERANDS (list of keyword operands)	6
ACCOUNT, EDIT, OPERATOR, OUTPUT, and TEST modes	
HELP	2
HELP subcommandname	7 8 9
HELP subcommandname ALL	7 8 9
HELP subcommandname FUNCTION	7
HELP subcommandname SYNTAX	8
HELP subcommandname OPERANDS	9
HELP subcommandname OPERANDS (list of keyword operands)	10

Figure 12. Information Available Through the HELP Command

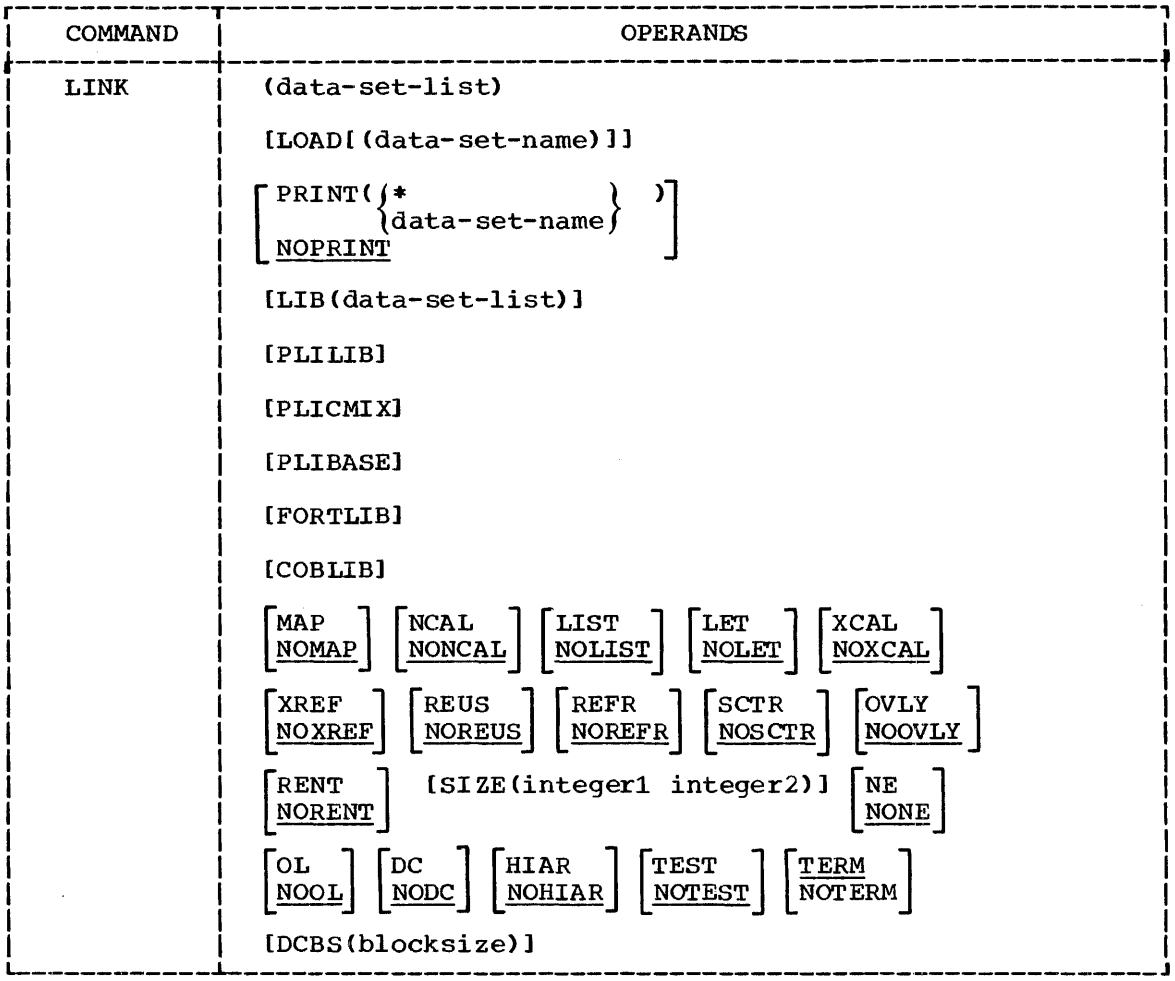
LINK Command

Use the LINK command to invoke the linkage editor service program. Basically, the linkage editor converts one or more object modules (the output modules from compilers) into a load module that is suitable for execution. In doing this, the linkage editor changes all symbolic addresses in the object modules into relative addresses. You can find a complete description of the functions of the linkage editor in the publication IBM System/360 Operating System: Linkage Editor and Loader, GC28-6538.

The linkage editor provides a great deal of information to help you test and debug a program. This information includes a cross-reference table and a map of the module that identifies the location of control sections, entry points, and addresses. You can have this information listed at your terminal or saved in a data set on some device.

You can specify all the linkage editor options explicitly or you can accept the default values. The default values are satisfactory for most uses. By accepting the default values, you simplify the use of the LINK command.

If the module that you want to process has a simple structure (that is, it is self contained and does not pass control to other modules) and you do not require the extensive listings produced by the linkage editor and you do not want a load module, you may want to use the LOADGO command as an alternative to the LINK command.



(data-set-list)
specifies the names of one or more data sets containing your object modules and/or linkage editor control statements. (See the data set naming conventions). The specified data sets will be concatenated within the output load module in the sequence that they are included in this operand. If there is only a single name in the data-set-list, parentheses are not required unless the single name is a member name of a partitioned data set; then, two pairs of parentheses are required, as in:

```
LINK((PARTS))
```

You may substitute an asterisk (*) for a data set name to indicate that you will enter control statements from your terminal. The system will prompt you to enter the control statements. A null line indicates the end of your control statements. The publication IBM System/360 Operating System: Linkage Editor and Loader, GC28-6538, contains a description of the control statements.

LOAD(data-set-name)
specifies the name of the partitioned data set that will contain the load module after processing by the linkage editor (see the data set naming conventions). If you omit this operand, the system will generate a name according to the data set naming conventions.

PRINT(data-set-name or *)
specifies that linkage editor listings are to be produced and placed in the specified data set. When you omit the data set name, the data set that is generated is named according to the data set naming conventions. You may substitute an asterisk (*) for the data set name if you want to have the listings displayed at your terminal. This is the default value if you specify the LIST, MAP, or XREF operand.

NOPRINT
specifies that no linkage editor listings are to be produced. This operand causes the MAP, XREF, and LIST options to become invalid. This is the default value if both PRINT and NOPRINT are omitted, and you do not use the LIST, MAP, or XREF operand.

LIB(data-set-list)
specifies one or more names of library data sets to be searched by the linkage editor to locate load modules referred to by the module being processed (that is, to resolve external references). (See the data set naming conventions.) When you specify more than one name, the names must be separated by a valid delimiter.

PLILIB
specifies that the partitioned data set named SYS1.PL1LIB is to be searched by the linkage editor to locate load modules that are referred to by the module being processed.

PLIBASE
specifies that the partitioned data set named SYS1.PLIBASE is to be searched to locate load modules referred to by the module being processed.

PLICMIX
specifies that the partitioned data set named SYS1.PLICMIX is to be searched to locate load modules referred to by the module being processed.

FORTLIB
specifies that the partitioned data set named SYS1.FORTLIB is to be searched by the linkage editor to locate load modules referred to by the module being processed.

COBLIB
specifies that the partitioned data set named SYS1.COBLIB is to be searched by the linkage editor to locate load modules referred to by the module being processed.

MAP
specifies that the PRINT data set is to contain a map of the output module consisting of the control sections, the entry names, and (for overlay structures) the segment number.

NOMAP
specifies that a map of the output module is not to be listed. This is the default value if both MAP and NOMAP are omitted.

NCAL
specifies that the automatic library call mechanism is not to be invoked to locate the modules that are referred to by the module being processed when the object module refers to other load modules.

NONCAL
specifies that the modules referred to by the module being processed are to be located by the automatic library call mechanism when the object module refers to other load modules. This is the default value if both NCAL and NONCAL are omitted.

LIST
specifies that a list of all linkage editor control statements is to be placed in the PRINT data set.

NOLIST
specifies that a listing of linkage editor control statements is not to be produced. This is the default value if both LIST and NOLIST are omitted.

LET
specifies that the output module is permitted to be marked as executable even though a severity 2 error is found (a severity 2 error indicates that execution of the output module may be impossible).

NOLET
specifies that the output module be marked non-executable when a severity 2 error is found. This is the default value if both LET and NOLET are omitted.

XCAL
specifies that the output module is permitted to be marked as executable even though an exclusive call has been made between segments of an overlay structure. Because the segment issuing an exclusive call is overlaid, a return from the requested segment can be made only by another exclusive call or a branch.

NOXCAL
specifies that both valid and invalid exclusive calls will be marked as errors. This is the default value if both XCAL and NOXCAL are omitted.

XREF
specifies that a cross-reference table is to be placed on the PRINT data set. The table includes the module map and a list of all address constants referring to other control sections. Since the XREF operand includes a module map, both XREF and MAP cannot be specified for a particular LINK command.

NOXREF
specifies that a cross-reference listing is not to be produced. This is the default value if both XREF and NOXREF are omitted.

REUS
specifies that the load module is to be marked serially reusable if the input load module was reenterable or serially reusable. The RENT and REUS operand are mutually exclusive. The REUS operand must not be specified if the OVLY or TEST operands are specified.

NOREUS
specifies that the load module is not to be marked reusable. This the default value if both REUS and NOREUS are omitted.

REFR
specifies that the load module is to be marked refreshable if the input load module was refreshable and the OVLY operand was not specified.

NOREFR
specifies that the load module is not to be marked refreshable. This is the default value if both REFR and NOREFR are omitted.

SCTR
specifies that the load module created by the linkage editor can be either scatter loaded or block loaded. If you specify SCTR, do not specify OVLY.

NOSCTR
specifies that scatter loading is not permitted. This is the default value if both SCTR and NOSCTR are omitted.

OVLY
specifies that the load module is an overlay structure and is therefore suitable for block loading only. If you specify OVLY, do not specify SCTR.

NOOVLY
specifies that the load module is not an overlay structure. This is the default value if both OVLY and NOOVLY are omitted.

RENT
specifies that the load module is marked reenterable provided the input load module was reenterable and that the OVLY operand was not specified.

NORENT
specifies that the load module is not marked reenterable. This is the default value if both RENT and NORENT are omitted.

SIZE(integer1,integer2)
specifies the amount of main storage to be used by the linkage editor. The first integer (integer1) indicates the maximum allowable number of bytes. Integer2 indicates the number of bytes to be used as the load module buffer, which is the main storage area used to contain input and output data. If this operand is omitted, SIZE defaults to the size specified at system generation (SYSGEN).

NE
specifies that the output load module cannot be processed again by the linkage editor or loader. The linkage editor will not create an external symbol dictionary. If you specify either MAP or XREF, this operand is invalid.

NONE
specifies that the output load module can be processed again by the linkage editor and loader and that an external symbol dictionary is present. This is the default value if both NE and NONE are omitted.

OL
specifies that the output load module can be brought into main storage only by the LOAD macro instruction.

NOOL
specifies that the load module is not restricted to the use of the LOAD macro instruction for loading into main storage. This is the default value if both OL and NOOL are omitted.

DC
specifies that the output module can be reprocessed by the linkage editor (E).

NODC
specifies that the load module cannot be reprocessed by the linkage editor (E). This is the default value if both DC and NODC are omitted.

HIAR
specifies that the control sections within the output module are to be marked for loading into either processor storage or IBM 2361 core storage. The linkage editor control statement HIARCHY assigns the appropriate hierarchy to the control sections. When you specify HIAR, the load module is marked suitable for scatter loading.

NOHIAR
specifies that no hierarchy assignments are to be made to the output load module. This is the default value if both HIAR and NOHIAR are omitted.

TEST
specifies that the symbol tables created by the assembler and contained in the input modules are to be placed into the output module.

NOTEST
specifies that no symbol table is to be retained in the output load module. This is the default value if both TEST and NOTEST are omitted.

TERM
specifies that you want error messages directed to your terminal as well as to the PRINT data set. This is the default value if both TERM and NOTERM are omitted.

NOTERM
specifies that you want error messages directed only to the PRINT data set and not to your terminal.

DCBS(blocksize)
specifies the blocksize of the records contained in the output load module "blocksize" must be an integer.

Example 1

Operation: Combine three object modules into a single load module.

Known: The names of the object modules in the sequence that the modules must be in..... DEPT03.GSALESA.OBJ
DEPT03.GSALESB.OBJ
DEPT03.NSALES.OBJ

You want all of the linkage editor listings to be produced and directed to your terminal.

The name for the output load module..... DEPT03.SALESRPT.LOAD(TEMPNAME)

```
LINK  (GSALESA,GSALESB,NSALES) LOAD(SALESRPT) PRINT(*)
XREF  LIST
```

Example 2

Operation: Create a load module from an object module, an existing load module, and a standard processor library.

Known: The name of the object module..... XRDJA3.M33THRUS.OBJ

The name of the existing load module. XRDJA3.M33PAYLD.LOAD(MOD1)

The name of the standard processor library used for resolving external references in the object module..... SYS1.PLILIB

The name of the output load module... XRDJA3.M33PERFO.LOAD(MOD2)

```
link (m33thrus,*) load(m33perfo(mod2)) print(*) plilib map list
```

Choosing ld2 as a filename to be associated with the existing load module, the listing at your terminal will be:

```
allocate dataset(m33payld.load) file(ld2)
link (m33thrus,*) load(m33perfo(mod2)) print(*) plilib map list
IKJ76080A  ENTER CONTROL STATEMENTS
 include ld2(mod1)
(null line)
IKJ76111I  END OF CONTROL STATEMENTS
```

Use the LISTALC command to obtain a list containing both the names of the data sets allocated by you and the names of the data sets temporarily allocated by previous TSO command processors. Also, this command specifies the number of data sets that the system will allow to be allocated to you dynamically. Included in the number of data sets that the system will allow a user to allocate dynamically, are data sets that had been previously allocated for temporary use by a command processor.

COMMAND	OPERANDS
{LISTALC / LISTA}	[STATUS] [HISTORY] [MEMBERS] [SYSNAMES]

STATUS
specifies that you want information about the status of each data set. This operand provides you with:

- The data definition name (DDNAME) for the data set.
- The scheduled and conditional dispositions of the data set. The keywords denoting the dispositions are CATLG, DELETE, KEEP and UNCATLG. The scheduled disposition is the normal disposition and precedes the conditional disposition when listed. The conditional disposition takes effect if an abnormal termination occurs. CATLG means that the data set is retained and its name is in the system catalog. DELETE means that references to the data set are to be removed from the system and the space occupied by the data set is to be released. KEEP means that the data set is to be retained. UNCATLG means that the data set name is removed from the catalog but the data set is retained.

HISTORY
specifies that you want to obtain information about the history of each data set. This operand provides you with:

- The creation date.
- The expiration date.
  Note: All data sets created by dynamic allocation will have creation and expiration dates of 00/00/00.
- An indication as to whether or not the data set has password protection.
- The data set organization (DSORG). The listing will contain:

  PS for sequential
  PO for partitioned
  IS for indexed sequential
  DA for direct access
  ** for unspecified
  ?? for any other specification

MEMBERS
specifies that you want to obtain the library member names of each partitioned data set having your user's identification as the leftmost qualifier of the data set name. Aliases will be included.

SYSNAMES
specifies that you want to obtain the fully qualified names of data sets having system-generated names.

Example 1

Operation: Obtain a list of the names of all the data sets allocated to you.

```
LISTALC
```

Example 2

Operation: Obtain a list of the names of all the data sets allocated to you. At the same time obtain the creation date, the expiration date, password protection, and data set organization for each data set allocated to you.

```
LISTA HISTORY
```

Example 3

Operation: Obtain all available information about the data sets allocated to you.

```
lista members history status sysnames
```

The output at your terminal will be similar to the following listing:

```
listalc mem status sysnames history

--DSORG--CREATED--EXPIRES---SECURITY---DDNAME---DISP

RRED95.ASM
  PS      00/00/00 00/00/00  WRITE       EDTDUMY1 KEEP

RRED95.EXAMPLE
  PO      00/00/00 00/00/00  PROTECTED   EDTDUMY2 KEEP,KEEP

--MEMBERS--
  MEMBER1
  MEMBER2

SYS70140.T174803.RV000.TSOSPEDT.R0000001

  **      00/00/00 00/00/00  NONE        SYSUT1   DELETE

3 DATA SETS CAN BE ALLOCATED DYNAMICALLY
  EDTDUMY3
  SYSIN
  SYSPRINT

READY
```

Use the LISTBC command to obtain a listing of the contents of the SYS1.BRODCAST data set. The SYS1.BRODCAST data set contains messages of general interest (NOTICES) that are sent from the system to all terminals and messages directed to a particular user (MAIL). The system deletes MAIL messages from the data set after they have been sent. NOTICES must be deleted explicitly by the operator.

COMMAND	OPERANDS
{LISTBC / LISTB}	[MAIL / NOMAIL] [NOTICES / NONOTICES]

MAIL
specifies that you want to receive the messages from the broadcast data set that are intended specifically for you. This is the default value if both MAIL and NOMAIL are omitted.

NOMAIL
specifies that you do not want to receive messages intended specifically for you.

NOTICES
specifies that you want to receive the messages from the broadcast data set that are intended for all users. This is the default value if both NOTICES and NONOTICES are omitted.

NONOTICES
specifies that you do not want to receive the messages that are intended for all users.

Example 1

Operation: Specify that you want receive all messages.

```
LISTBC
```

Example 2

Operation: Specify that you want to receive only the messages intended for all terminal users.

```
listbc nomail
```

Use the LISTCAT command to obtain a list of the names of your cataloged data sets.

The system catalog is a data set that contains the location of other data sets. The catalog is organized into levels of indexes that connect the data set names to corresponding locations (volumes and data set sequence numbers). Each qualifier in the data set name (see the data set naming conventions) corresponds to one of the indexes in the catalog. For instance, suppose that a data set named D58JCD.GSCORE.DATA is cataloged. The catalog has a master index that contains D58JCD as an entry. This entry includes the location of an index named D58JCD. The index named D58JCD contains GSCORE as an entry that includes the location of an index named GSCORE. The index named GSCORE contains DATA as an entry that includes the location of the data set.

The LISTCAT command, when entered with no operands, produces a list of all cataloged data sets that have your user identification as the leftmost qualifier. You can request a partial, more specific list by identifying the index level that you want to have listed. You can specify any index level in the catalog.

COMMAND	OPERANDS
{LISTCAT / LISTC}	[HISTORY] [MEMBERS] [VOLUMES] [LEVEL(index)]

HISTORY
specifies that you want information about the history of each data set. This operand provides you with:

- The creation date.
- The expiration date.
  Note: All data sets created by dynamic allocation will have creation and expiration dates of 00/00/00.
- An indication as to whether or not the data set has password protection.
- The data set organization (DSORG).

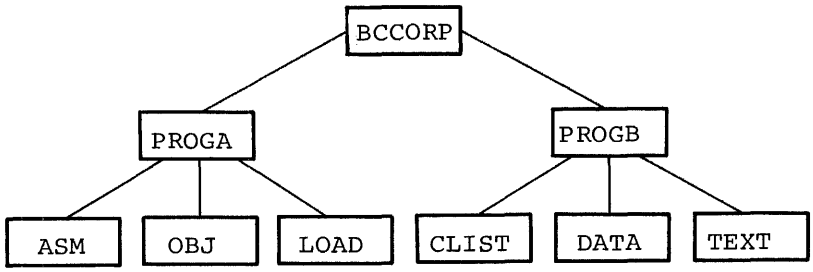
  The listing will contain:

  PS for sequential
  PO for partitioned
  IS for indexed sequential
  DA for direct access
  ** for unspecified
  ?? for any other specification

MEMBERS
specifies that you want a list of names for the members of each partitioned data set. Alias names will be included.

VOLUMES
specifies that you want the volume identification (VOLID) for each volume on which the data sets reside. A volume may be a reel of tape, a disk pack, a bin in a data cell, or a drum.

LEVEL(index)
specifies that you want the names of only a portion of the cataloged data sets. You indicate an index level by including one or more data set name qualifiers for 'index'. All data sets at an index level that is lower than the one that you indicate will be listed. For instance, if you have an index structure such as:



and you specify LEVEL(BCCORP.PROGA), you will receive:

```
ASM (meaning  BCCORP.PROGA.ASM)
OBJ (meaning  BCCORP.PROGA.OBJ)
LOAD (meaning BCCORP.PROGA.LOAD)
```

The specified index must begin with the highest level of qualification (for example, your user identification, or SYS1). You may also include one asterisk in your specified index qualification. The asterisk indicates that all qualifiers corresponding to the position of the asterisk are to be considered as if each was specified explicitly. The asterisk must not be placed at the highest or lowest level.

Example 1

Operation: List the names of all of your cataloged data sets.

```
LISTCAT
```

Example 2

Operation: List the names of all of your cataloged data sets; include their history and the volumes that they reside on.

```
LISTCAT HISTORY VOLUMES
```

The listing produced at your terminal will appear similar to the following simulated listing.

```
READY

listcat history volumes
```

```
--DSORG--CREATED---EXPIRES---SECURITY

CLIST.FLOWCHRT
  PS      07/11/66   09/14/70   NONE

--VOLUMES--
  D58LIB

XERPT.TEXT
  PS      00/00/00   00/00/00   NONE

--VOLUMES--
  D58LIB

READY
```

Example 3

Operation: List the names, history and volumes of a particular selection of your cataloged data sets.

The names of your data sets..... RCHD58.FLOW1.FORT
RCHD58.FLOW2.FORT
RCHD58.FLOW3.FORT

```
LISTCAT LEVEL(RCHD58.*.FORT) HISTORY VOLUMES
```

The listing produced at your terminal will appear similar to the following simulated listing.

```
READY

listcat level(rchd58.*.fort) volumes history

--DSORG--CREATED---EXPIRES---SECURITY

RCHD58.FLOW1.FORT
  PS      00/00/00   00/00/00   NONE

--VOLUMES--
  D58CAT

RCHD58.FLOW2.FORT
  PS      00/00/00   00/00/00   PROTECTED

--VOLUMES--
  D58CAT

RCHD58.FLOW3.FORT
  PS      00/00/00   00/00/00   WRITE

--VOLUMES--
  D58CAT

READY
```

Use the LISTDS command to have the attributes of specific data sets displayed at your terminal. You can obtain:

- The volume identification (VOLID) of the volume on which the data set resides. A volume may be a disk pack, a bin in a data cell, or a drum.
- The record format (RECFM), the logical record length (LRECL), and the blocksize (BLKSIZE) of the data set.
- The data set organization (DSORG).

  The data set organization is indicated as follows:

  PS for sequential
  PO for partitioned
  IS for indexed sequential
  DA for direct access
  ** for unspecified
  ?? for any other specification

- Directory information for members of partitioned data sets if you specify the data set name in the form data set name(membername).
- Creation date, expiration date, and security attributes
- File name and disposition.
- Data set control blocks (DSCB).

COMMAND	OPERANDS
{LISTDS / LISTD}	(data-set-list) [STATUS] [HISTORY] [MEMBERS] [LABEL]

(data-set-list)
specifies one or more data set names (see the data set naming conventions). This operand identifies the data sets that you want to know more about. Each data set specified must be currently allocated or available from the catalog, and must reside on a currently active volume.

STATUS
specifies that you want the following additional information:

- The data definition (DD) name DDNAME currently associated with the data set.
- The currently scheduled data set disposition and the conditional disposition. The keywords denoting the dispositions are CATLG, DELETE, KEEP, and UNCATLG. The scheduled disposition is the normal disposition and precedes the conditional disposition when listed. The conditional disposition takes effect if an abnormal

termination occurs. CATLG means that the data set is cataloged. DELETE means that the data set is to be removed. KEEP means that the data set is to be retained. UNCATLG means that the name is removed from the catalog but the data set is retained.

HISTORY
specifies that you want to obtain the creation and expiration dates for the specified data sets (all data set created by dynamic allocation will have creation and expiration dates of 00/00/00), and to find out whether or not the data sets are password protected.

MEMBERS
specifies that you want a list of all the members of a partitioned data set including any aliases.

LABEL
specifies that you want to have the entire data set control block (DSCB) listed at your terminal. This operand is applicable only to direct access data sets. The listing will be in hexadecimal notation.

Example 1

Operation: List the basic attributes of a particular data set.

Known: The data set name..... RCHD95.CIR.OBJ

```
LISTDS CIR
```

The listing produced at your terminal will be similar to the listing shown below.

```
READY

listds cir

RCHD95.CIR.OBJ
--RECFM-LRECL-BLKSIZE-DSORG
  FB    80     80       PS

--VOLUMES--
  D95LIB

READY
```

Example 2

Operation: List the basic attributes and the DSCBs for a particular data set.

Known: The data set name..... RCHD95.IKJEHDS1.LOAD

```
listd ikjehds1 label
```

Use the LOADGO command to load a compiled or assembled program into main storage and begin execution.

The LOADGO command will load object modules produced by a compiler or assembler, and load modules produced by the linkage editor. (If you want to load and execute a single load module, the CALL command is more efficient.) The LOADGO command will also search a call library (SYSLIB) or a resident link pack area, or both, to resolve external references.

The LOADGO command invokes the system loader to accomplish this function. The loader combines basic editing and loading services of the linkage editor and program fetch in one job step (see the publication IBM System/360 Operating System: linkage Editor and Loader, GC28-6538). Therefore, the load function is equivalent to the link edit and go function.

The LOADGO command does not produce load modules for program libraries, and it does not process linkage editor control statements such as INCLUDE, NAME, OVERLAY, etc.

COMMAND	OPERANDS
{LOADGO / LOAD}	(data-set-list) ['parameters']
	[PRINT ({* / data-set-name}) / NOPRINT]
	[LIB(data-set-list)]
	[PLILIB] [PLIBASE] [PLICMIX] [FORTLIB] [COBLIB]
	[TERM / NOTERM] [RES / NORES] [MAP / NOMAP] [CALL / NOCALL] [LET / NOLET]
	[SIZE(integer)]
	[EP(entry-name)]
	[NAME(program-name)]

(data-set-list)
specifies the names of one or more object modules and/or load modules to be loaded and executed. The names may be data set names, names of members of partitioned data sets, or both (see the data set naming conventions). When you specify more than one name, the names must be enclosed within parentheses and separated from each other by a standard delimiter (blank or comma).

'parameters'
specifies any parameters that you want to pass to the program to be executed.

PRINT(data-set-name or *)
specifies the name of the data set that is to contain the listings produced by the LOADGO command. If you omit the data set name, the generated data set will be named according to the data set naming conventions. You may substitute an asterisk (*) for the data set name if you want to have the listings displayed at your terminal. This is the default if you specify the MAP operand.

NOPRINT
specifies that no listings are to be produced. This operand negates the MAP operand. This is the default value if both PRINT and NOPRINT are omitted, and you do not use the MAP operand.

TERM
specifies that you want any error messages directed to your terminal as well as the PRINT data set. This is the default value if both TERM and NOTERM are omitted.

NOTERM
specifies that you want any error messages directed only to the PRINT data set.

LIB(data set list)
specifies the names of one or more library data sets that are to be searched to find modules referred to by the module being processed (that is, to resolve external references).

PLILIB
specifies that the partitioned data set named SYS1.PL1LIB is to be searched to locate load modules referred to by the module being processed.

PLIBASE
specifies that the partitioned data set named SYS1.PLIBASE is to be searched to locate load modules referred to by the module being processed.

PLICMIX
specifies that the partitioned data set named SYS1.PLICMIX is to be searched to locate load modules referred to by the module being processed.

COBLIB
specifies that the partitioned data set named SYS1.COBLIB is to be searched to locate load modules referred to by the module being processed.

FORTLIB
specifies that the partitioned data set named SYS1.FORTLIB is to be searched to locate load modules referred to by the module being processed.

RES
specifies that the link pack area is to be searched for load modules (referred to by the module being processed) before the specified libraries are searched. This is the default value if both RES and NORES are omitted. If you specify the NOCALL operand the RES operand is invalid.

NORES
specifies that the link pack area is not to be searched to locate modules referred to by the module being processed.

MAP
specifies that a list of external names and their absolute storage addresses are to be placed on the PRINT data set. This operand is ignored when NOPRINT is specified.

NOMAP
specifies that external names and addresses are not to be contained in the PRINT data set. This is the default value if both MAP and NOMAP are omitted.

CALL
specifies that the data set specified in the LIB operand is to be searched to locate load modules referred to by the module being processed. This is the default value if both CALL and NOCALL are omitted.

NOCALL
specifies that the data set specified by the LIB operand will not be searched to locate modules that are referred to by the module being processed. The RES operand is invalid when you specify this operand.

LET
specifies that execution is to be attempted even if a severity 2 error should occur. (A severity 2 error indicates that execution may be impossible.)

NOLET
specifies that execution is not to be attempted if a severity 2 error should occur. This is the default value if both LET and NOLET are omitted.

SIZE(integer)
specifies the size, in bytes, of dynamic main storage that can be used by the loader. If this operand is not specified, then the size defaults to the size specified at System Generation (SYSGEN).

EP(entry-name)
specifies the external name for the entry point to the loaded program. You must specify this operand if the entry point of the loaded program is in a load module.

NAME(program-name)
specifies the name that you want assigned to the loaded program.

Example 1

Operation: Load and execute an object module.

Known: The name of the data set..... SHEPD58.CSINE.OBJ

```
LOADGO  CSINE  PRINT(*)
```

Example 2

Operation: Combine an object module and a load module, and then load and execute them.

Known: The name of the data set containing the object module..... LARK.HINDSITE.OBJ
The name of the data set containing the load module..... LARK.THERMOS.LOAD(COLD)

```
LOAD  (HINDSITE  THERMOS(COLD)) PRINT(*) LIB('SYS1.SORTLIB')
NORES MAP SIZE(44K) EP(START23) NAME(THERMSIT)
```

LOGOFF Command

Use the LOGOFF command to terminate your terminal session.

Before you enter the LOGOFF command, you should use the EDIT command's SAVE subcommand to store the data sets that you want to save. When you enter the LOGOFF command, the system frees all the data sets allocated to you; data remaining in main storage will be lost.

Note: If you intend to enter the LOGON command immediately and continue processing against a different account number you do not enter LOGOFF. Instead, you can just enter the LOGON command as you would enter any other command.

COMMAND	OPERAND
LOGOFF	

Example 1

Operation: Terminate your terminal session.

```
logoff
```

LOGON Command

Use the LOGON command to initiate a terminal session. Before you can use the LOGON command, your installation must provide you with certain basic information.

- Your user identification (1-7 characters) and, if required by your installation, a password (1-8 alphameric characters).
- An account number (may or may not be required for your installation).
- A procedure name (may or may not be required for your installation).

You must supply this information to the system by using the LOGON command and operands. The information that you enter is used by the system to start and control your time sharing terminal session.

You can also use the operands to specify whether or not you want to receive messages from the system or other users.

COMMAND	OPERANDS
LOGON	user-identity[/password]
	[ACCT(account)]
	[PROC(procedure)]
	[SIZE(integer)]
	[NOTICES / NONOTICES]
	[MAIL / NOMAIL]

user-identity and password
: specifies your user identification and, if required, a valid password. You user identification must be separated from the password by a slash (/) and, optionally, one or more standard delimiters (tab, blank, or comma). Your identification and password must match the identification contained in the system's User Attribute Data Set (UADS). If you omit any part of this operand, the system will prompt you to complete the operand. (Printing is suppressed for some types of terminals when you respond to a prompt for a password.)

ACCT(account)
: specifies the account number required by your installation. If the UADS contains only one account number for the password that you specify, this operand is not required. If the account number is required and you omit it, the system will prompt you for it.

  For TSO, an account number must not exceed 40 characters, and must not contain a blank, tab, quotation mark, apostrophe, semicolon, comma, or line control character. Right parentheses are permissible only when left parentheses balance them somewhere in the account number.

PROC(procedure-name)
specifies the name of a cataloged procedure containing the Job Control Language (JCL) needed to initiate your session.

SIZE(integer)
specifies the size of the main storage region, in units of 1024 bytes, that you want allocated to your job. The UADS contains a default value for your region size if you omit this operand. The UADS also contains a value for the maximum region size that you will be allowed. This operand will be rejected if you specify a region size exceeding the maximum region size allowed by the UADS (in this case, the UADS value MAXSIZE will be used).

NOTICES
specifies that messages intended for all terminal users are to be listed at your terminal during LOGON processing. This is the default value if both NOTICES and NONOTICES are omitted.

NONOTICES
specifies that you do not want to receive the messages intended for all users.

MAIL
specifies that you want messages intended specifically for you to be displayed at your terminal. This is the default value if both MAIL and NOMAIL are omitted.

NOMAIL
specifies that you do not want to receive messages intended specifically for you.

Example 1

Operation: Initiate a terminal session.

Known: Your user identification and password..... AJKD58/23XA$MBT
Your installation does not require an account number or procedure name for LOGON.

```
LOGON AJKD58/23XA$MBT
```

Example 2

Operation: Initiate a terminal session.

Known: Your user identification and password..... HEUS951/MO@
Your account number..... 288104
The name of a cataloged procedure..... TS951
You do not want to receive messages.
Your main storage space requirement..... 90K bytes

```
LOGON HEUS951/MO@ ACCT(288104) PROC(TS951) SIZE(90)NONOTICES NOMAIL
```

Use the OPERATOR command (along with its subcommands) to regulate and maintain TSO from a terminal.

The OPERATOR command is fully supported only for terminals which have the transmit interruption capability, that is, this command is supported only for those terminals for which the BREAK operand of the TERMINAL command is valid.

This command may be used only by personnel who have been given the authority to do so by the installation management. The authority to use OPERATOR is normally given to personnel responsible for system operation, and is recorded in the User Attribute Data Set (see the ACCOUNT command).

COMMAND	OPERANDS
{OPERATOR {OPER	

THE OPERATOR COMMAND

The OPERATOR command, through the use of its eight subcommands, allows the terminal user to control TSO as follows:

Subcommand	Function Performed
CANCEL	Cancel a terminal session or a background job submitted through TSO.
DISPLAY	Display the number of users in a region, the number of batch jobs submitted via the SUBMIT command, the TSO job messages that are awaiting a reply from the system operator, and the number of active terminals.
END	Terminate operation of the OPERATOR command (thereby removing the user's terminal from OPERATOR mode).
HELP	Get a list of the subcommands of the OPERATOR command, along with the function, syntax, and operands of the subcommands.
MODIFY	Modify TSO options that were specified when the system was generated or when time-sharing was initiated.
MONITOR	Monitor both terminal and background job activities within the system. Informational messages will be displayed.
SEND	Send or receive a message to or from other terminal users.
STOPMN	Terminate the monitoring operations of the MONITOR subcommand; the display of status information at the user's terminal will be stopped.

FORMAT

The OPERATOR command and its eight subcommands have a format that is compatible with the MVT system operator commands of the System/360 Operating System. The similarities between the TSO OPERATOR commands and the MVT system operator commands are shown in Figure 13. The MVT commands are documented in: IBM System/360 Operating System: Operator's Reference, GC28-6691.

SYNTAX

When using the OPERATOR subcommands the following should be noted:

- Blanks are the only valid characters allowed between a subcommand and its operand.
- One comma is the only valid character allowed between operands.
- Operand length, including delimiters, is limited to 124 characters.

TSO Command Language MVT Command Language (Without TSO)

Terminal User OPERATOR		System Operator	
Subcommand	Operands	Command	Operands
CANCEL		CANCEL	
	jobname		jobname
	DUMP		DUMP
	ALL		ALL
	IN=class		IN=class
	OUT=class		OUT=class
	unit-address		UNITADDR
	identifier		identifier
	U=userid		
DISPLAY		DISPLAY	
	jobname		jobname
	A		A
	T		T
	N		N
	Q		Q
	list		list
	R		R
	USER[=NMBR]		
			U
			C,K
			CONSOLES
	Q=		Q=
	N=		N=
	SQA		SQA

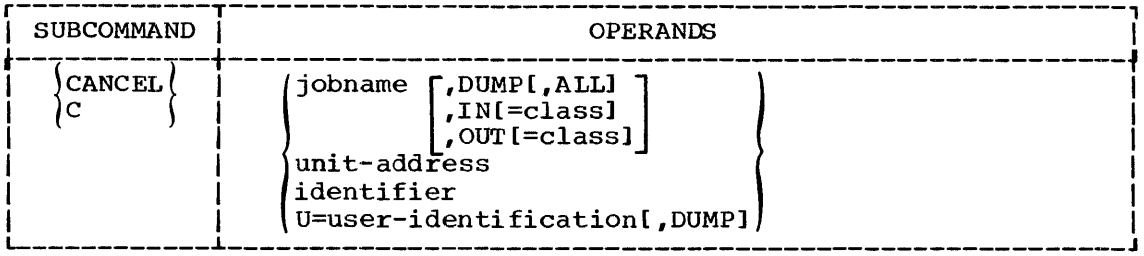
Figure 13. Relationships Between the TSO OPERATOR Subcommands and the MVT (non-TSO) Operator Commands (Part 1 of 2)

TSO Command Language		MVT Command Language (Without TSO)	
Terminal User OPERATOR		System Operator	
Subcommand	Operands	Command	Operands
MODIFY	procedure.identification USERS=nmbr SUBMIT=Queuesize REGSIZE(n)=(nnnnnK,xxxxxK) DRIVER=(parameters) HOLD=(region-list) SMF=	MODIFY	procname.identifier CLASS=classnames PAUSE=FORMS PAUSE=DATASET JOBCLASS=jjj OUTCLASS=s 'job parameters'
MONITOR	A (same as DISPLAY A) SESS[,T] STATUS JOBNAMES[,T] SPACE DSNAME	MONITOR	 T STATUS JOBNAMES SPACE DSNAME
STOPMN	 JOBNAMES SPACE DSNAME STATUS SESS	STOPMN	 procname.identifier JOBNAMES SPACE DSNAME STATUS
SEND	'text' USER=(userid list) ALL NOW LOGON message number,DELETE message number,LIST LIST	None	
HELP	subcommand-name FUNCTION SYNTAX OPERANDS ALL	None	
END		None	

Figure 13. Relationships Between the TSO OPERATOR Subcommands and the MVT (non-TSO) Operator Commands (Part 2 of 2)

CANCEL Subcommand of OPERATOR

Use the CANCEL subcommand to terminate the current activities of a terminal user or a job submitted for conventional batch processing. When you use the CANCEL command to terminate a terminal session, accounting information will be presented to the user. The syntax for this subcommand is the same as the syntax for the MVT operator commands.



jobname
is the name of the job that you want to cancel.

DUMP
specifies that an abnormal-end-of-job storage dump will be taken if a step of the job is being executed when you enter the command. The dump will be printed on the system output device.

ALL
specifies that all the input and output for the job is to be canceled.

IN=class
specifies that the system is to search for the job on the input queue indicated by "class". If you omit "class", all input queues will be searched.

OUT=class
specifies that the system is to search for the job on the output queue indicated by "class". If you omit "class", all output queues will be searched.

Note: If neither the IN or OUT parameter is used the system will search all the input queues and the hold queue for the job.

unit address
specifies the address of an I/O device. The system will stop the output currently being written on the device.

identifier
specifies the identifier of a system task to be terminated during allocation. You cannot cancel a system task that is not associated with a unit (device).

This operand can be the identifier used in a START command issued by the system's console operator or it can be a unit type (such as 1403 or 2311) associated with a unit address or a procedure used in a START command.

U=user identification
specifies the user identification for a user whose terminal session is to be terminated by the CANCEL command.

Note: Use the 'CANCEL U=userid' format for canceling time sharing jobs only and the 'CANCEL jobname' format for canceling conventional batch jobs only.

Example 1

Operation: Terminate a user's terminal session.

Known: The user's identification..... RCHTD36

```
C U=RCHTD36
```

Example 2

Operation: Cancel a job that has been submitted from a terminal for conventional batch processing and have a dump printed.

Known: The name of the job..... PAYROLL

```
cancel payroll,dump
```

Example 3

Operation: Cancel the output from a job that has been submitted from a terminal for conventional batch processing.

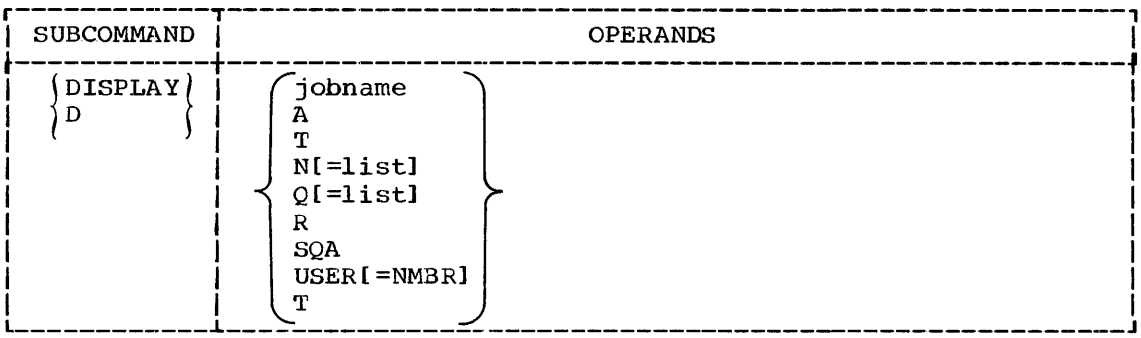
Known: The name of the job..... SUEG
The output class..... J

```
CANCEL SUEG,OUT=J
```

DISPLAY Subcommand of OPERATOR

Use the DISPLAY subcommand to obtain a listing of:

- The number of terminal users for each time sharing region.
- The number of conventional batch jobs awaiting execution that were submitted from a terminal by the SUBMIT command.
- The messages from time sharing jobs that are awaiting replies from an operator.
- The message indicating the status of the system queue area (SQA).
- The number of active terminals, the identification of each user, and the time sharing region being used by each user. The operands 'jobname', 'A', 'N', 'Q', 'T' and 'R' are also operands of the DISPLAY command of the System/360 Operating System. They are described in detail in the publication IBM System/360 Operating System: Operator's Reference, GC28-6691. The syntax for this subcommand is the same as the syntax for the MVT operator commands.



jobname
specifies the name of the job for which the following status information is to be displayed: job name; class; job priority; type of queue the job is in (JOB Q, HOLD Q, SOUT Q (SYSOUT queue), or BRDR): and the job's position in the queue.

The maximum length of a job name is eight characters. If your jobname is JOBNAMES, STATUS, T, A, R, Q, N, SPACE, DSNAME, SESS, USER, U, M, or CONSOLES it must be enclosed in parentheses.

A
specifies that you want the system to display information about all jobs and jobsteps that are recognized by the system as tasks (that is, those jobs and job steps that have one or more task control block (TCB)).

The information displayed for jobs in background regions includes the names of the job and job step associated with each task, the number of subordinate tasks operating within the same region of main storage, the beginning and end addresses of the region, and the amount of supervisor queue space used for system control blocks related to the main task. If rollout is included in the system, the display will indicate whether the region is borrowed or rolled out.

The information displayed for time-sharing regions includes TIME SHARING as the job name, the number of users for each region, the region number, the beginning and end addresses of the region, and the amount of local supervisor queue space used for system control blocks by the user's tasks.

N
specifies that you want a list of job names on the input, hold, output, BRDR, and ASB reader batching queues.

Q
specifies that you want a list of the number of entries on the input, hold, output, BRDR, and ASB reader batching queues.

list
specifies that you want information about specific queues. You can specify up to four of the following queues:

- Specific input work queue name (job class A through 0).
- SOUT (system output queues collectively).
- HOLD (system hold queue).
- BRDR (background reader queue).

R
specifies that you want a listing of messages that are awaiting a response from an operator.

SQA
specifies that you want information on the system queue area (SQA). You will receive a message at your terminal indicating the low and high boundaries of the SQA and the amount of free storage between them.

T
specifies that you want the time of day and the date.

USER[=NMBR]
indicates you want specific information about time sharing users. If you do not specify =NMBR, the number of active terminals, the identification of each user and the corresponding region number of each user will be displayed at your terminal. If you do specify =NMBR, only the number of active terminals will be displayed.

Example 1

Operation: Have the number of time sharing regions and the number of users for each region displayed at your terminal.

```
DISPLAY A
```

Example 2

Operation: Have the status of a particular job displayed at your terminal.

Known: The name of the job is..... RBTATT

```
display  rbtatt
```

Example 3

Operation: Obtain the user identification for each active terminal user.

```
d user
```

Use the END subcommand to terminate operation of the OPERATOR command. After entering the END subcommand, you may enter new commands.

SUBCOMMAND	OPERANDS
END	

HELP Subcommand of OPERATOR

Use the HELP subcommand to find out how to use OPERATOR and the OPERATOR subcommands. When you enter the HELP subcommand, the system responds by printing out explanatory information at your terminal. You may request:

- A list of available subcommands.
- An explanation of the fucntion, syntax, and operands of a specific subcommand.

The HELP subcommand actually causes the system to execute a function of the HELP command; therefore, you may consult the discussion of the HELP command if you desire more detailed information.

SUBCOMMAND	OPERANDS
{HELP / H}	[subcommand-name [FUNCTION / SYNTAX / OPERANDS(list-of-operands) / ALL]]

subcommand-name
specifies the subcommand that you want to have clarified. If you omit this operand, the system will display a list of OPERATOR subcommands.

FUNCTION
specifies that you want a description of the referenced subcommand's function.

SYNTAX
specifies that you want a definition of the proper syntax for the referenced subcommand.

OPERANDS(list of operands)
specifies that you want an explanation of the operands applicable to the referenced subcommand.

The list of operands specifies the particular keywords that you want to have explained. If you do not specify any keywords, all of the applicable keywords will be included. You must use one or more blanks or a comma as a delimiter between the keywords in the list.

ALL
specifies that you want a description of the function, the syntax, and the operands of the subcommand that you specified. This is the default value if no operands are specified.

Example 1

Operation: Have a list of available subcommands displayed at your terminal.

```
HELP
```

Example 2

Operation: Obtain all available information about a particular subcommand.

Known: The subcommand name..... MODIFY

```
H MODIFY
```

Example 3

Operation: Have a list of the operands for a particular subcommand displayed at your terminal.

Known: The subcommand name..... DISPLAY

```
h DISPLAY operands
```

Use the MODIFY subcommand to modify the time sharing options that were specified when the system was generated or when time sharing was initiated. The syntax used for this subcommand is the same as the syntax used for MVT operator commands.

SUBCOMMAND	OPERANDS
{MODIFY / F}	[procedure.]identification [,USERS=number] [,SUBMIT=(queuesize)] [,REGSIZE(n)=(nnnnnK,xxxxxK)...] [,DRIVER=(parameters)] [,HOLD=(region-list)] [,SMF=([OFF / OPT= 1 / 2] [,EXT={YES / NO}])]

procedure
specifies the name of the time sharing procedure that you want to modify. This name must be the same as the one that was used when the procedure was started with a START command issued by the console operator.

identification
specifies the identification of the system task used when the task was defined by a START command issued by the console operator.

USERS=number
indicates the number of users allowed for time sharing. The maximum number is determined at START time and cannot be exceeded by the MODIFY command.

SUBMIT=Queuesize
indicates the maximum number of logical tracks to be used for the queue for conventional batch jobs submitted by the SUBMIT command.

REGSIZE(n)=(nnnnnK,xxxxxK)
indicates the number and size of each time sharing region. 'n" is the region number (included on the informational messages from the DISPLAY command). You specify the size of the region in the form nnnnnK. If the size equals zero the region will be freed. You specify the local supervisor queue area (LSQA) to be added to the region in the form xxxxxK. The LSQA size must be smaller than the region size, but greater than zero. "nnnnn" and "xxxxx" are the number of contiguous 1024 byte areas you want. These numbers may range from one to five digits, but the sum cannot exceed 16382. The numbers should be specified as even numbers. (If you specify an odd number, the system treats it as the next higher even number). LSQA size must be smaller than the region size, but greater than zero. Anytime you use the REGSIZE operand, any users of that region will be logged off.

DRIVER=(parameters)
specifies a parameter list to be passed to the time sharing driver (a component of TSO). For instance, BACKGROUND=value is the only keyword that can be passed to the IBM supplied driver -- it indicates the percentage of system resource time quaranteed for conventional batch processing; however, different parameters may be supplied for user-written drivers.

HOLD=(region-list)
specifies that the time-sharing regions specified in "region-list" are not to be allocated for any new users. If you specify more than one region, then you must separate the regions specified with commas. If you specify only one region, the parentheses are not needed.

You may not specify HOLD and REGSIZE(n) for the same region in one MODIFY command. If you do, the system will request that you: specify the option you prefer, indicate that both keywords are to be ignored for this region, or cancel the MODIFY command.

SMF=(OFF or OPT=1 or OPT=2, EXT=YES or NO)
indicates which option of the System Management Function (SMF) is to be used for time sharing operations. OFF indicates that SMF is not to be used for time sharing operations. OPT=1 or 2 indicates an option of SMF that is to be used for time sharing operations. EXT indicates that exits to the installation routines are active.

Note: If duplicate keywords are entered in a MODIFY command, the right-most (last entered) keyword and parameters will determine system action. (n) is part of the REGSIZE(n) keyword; therefore, REGSIZE(1) and REGSIZE(2) are not considered duplicate keywords.

Example 1

Operation: Change the number of terminals allowed for time sharing operations.

Known: The existing allowable number..... 32
The new number..... 26

```
MODIFY TSO,USERS=26
```

Example 2

Operation: Change the maximum size of time sharing region number 3 from 70K to 100K, with 10K reserved for local supervisor queue area (LSQA).

```
f tso,regsize(3)=(100K,10K)
```

Example 3

Operation: Change the guaranteed background percentage of time to 60%.

```
F TSO,DRIVER=(BACKGROUND=60)
```

MONITOR Subcommand of OPERATOR

Use the MONITOR subcommand to monitor terminal activities and job activities within the system. Informational messages will be displayed. The content of the messages will pertain to the type of information indicated by the operand included with the MONITOR subcommand. The system will continue to issue these informational messages until halted by a STOPMN subcommand or until you terminate the OPERATOR command.

SUBCOMMAND	OPERANDS
{MONITOR / MN}	{A / SESS[,T] / STATUS / JOBNAMES[,T] / SPACE / DSNAME}

A
: specifies that you want the system to display information about all of the jobs and jobsteps that are recognized by the system as tasks. (Under the TSO OPERATOR command function, the MONITOR A subcommand produces the same results as the DISPLAY A subcommand. Time-interval updating of the display is not supported under TSO.)

SESS
: indicates that you are to be notified whenever any terminal session is initiated or terminated. The user's identification will be displayed at your terminal. If the session terminates abnormally, the user identification will appear in the diagnostic message; the message "user LOGGED OFF" will not appear if the session was canceled.

  If you specify the T operand, the system displays the time of day in addition to the users identification. The format of the time output is shown under the T operand description.

T
: specifies that you want the time of day to be displayed in the following format:

  hh.mm.ss

  The variables in this format are:

  hh - Hours (00-23)

  mm - Minutes (00-59)

  ss - Seconds (00-59)

  whenever one TSO user specifies this operand, all subsequent users of the MONITOR command will also receive the time at their terminals.

STATUS
specifies that you want the data set names and volume serial numbers of data sets with dispositions of KEEP, CATLG, or UNCATLG to be displayed whenever the data sets are freed.

JOBNAMES
specifies that you want the name of each job to be displayed both when the job starts and when it terminates, and that you want unit record allocation to be displayed when the job step starts. If a job terminates abnormally, the jobname will appear in the diagnostic message; the message 'jobname ENDED' will not appear.

If you specify the T operand with the JOBNAMES operand, the system displays the time of the day in addition to the jobnames. The format of the output is shown under the T operand description.

SPACE
specifies that you want the system to display, in demount messages, the available space on a direct access device.

DSNAME
specifies that you want the system to display, within the mount and K (keep) type demount messages, the name of the first non-temporary data set allocated to the volume to which the messages refer.

Example 1

Operation: Have the system notify you whenever a terminal session begins or ends.

```
MONITOR SESS
```

Example 2

Operation: Have displayed at your terminal the name of each job when the job starts and when it terminates. Also have the time displayed with the jobname.

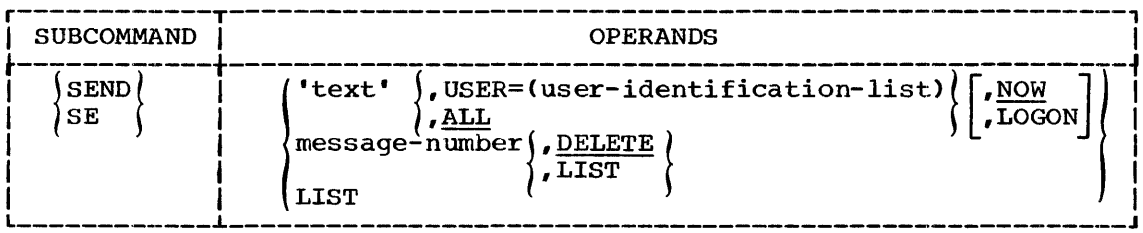
```
MN JOBNAMES,T
```

SEND Subcommand of OPERATOR

Use the SEND subcommand to send a message to any or all terminal users. A message may be sent to one or more terminal users by indicating the user identification of each recipient, or to all terminal users by not indicating specific user identifications. If the intended recipient is not logged on, the message can be retained within the system and presented automatically when the recipient logs on. You will be notified when the recipient of an immediate message is not logged on: the message will be deleted by the system.

Note: The maximum length of the SEND subcommand operands, including delimiters, is 124 characters.

The syntax for this subcommand is the same as the syntax for MVT operator commands.



'text'
specifies the message that you want to send. You must enclose the text of the message within apostrophes (single quotes). The maximum length of a message is 115 characters including blanks. The message must be contained on one line (you cannot continue a message on a second line). If you want a quotation mark printed in the message, you must enter two quotation marks.

USER=(user identification list)
specifies the user indentification of one or more terminal users who are to receive the message.

ALL
specifies that all terminal users are to receive the message. Terminal users who are currently using the system will receive the message immediately. This is the default value if both USER=(user identification list) and ALL are omitted.

NOW
specifies that the message is to be sent immediately. If the recipient is not logged on, you will be notified and the message will be deleted. This is the default value if NOW and LOGON are omitted.

LOGON
specifies that the message is to be sent immediately if the recipients are logged on and receiving messages. Otherwise, the message is to be retained in the SYS1.BRODCAST data set if:

a. You specify a user identification the message is retained in the "mail" section of the SYS1.BRODCAST data set and deleted by the system after it is sent to the intended user.

b. You specify "ALL", the message will be stored in the "notices" section of the SYS1.BRODCAST data set and retained there until the operator deletes it.

message number,DELETE
specifies the number of a notice in the SYS1.BRODCAST data set that you want to delete.

message number,LIST
specifies the number of a notice in the SYS1.BRODCAST data set that you want to have displayed at your terminal. Anytime you specify a message number without either the LIST or DELETE operand, the system assumes the default value and deletes the message.

LIST
specifies that you want to receive a listing of all the SEND notices retained in the system. The listing will be produced at your terminal. Each message will be preceded by a system-assigned number.

Example 1

Operation: Send a message to all terminal users currently logged on.

Known: The message:
TSO TO SHUT DOWN AT 9:55 P.M. EST 9/14/70

```
SEND 'TSO TO SHUT DOWN AT 9:55 P.M.  EST 9/14/70',ALL
```

Example 2

Operation: Send a message to two particular terminal users currently logged on.

Known: The user identifications..... T24
OTO

The message:
YOUR ACCT NO. INVALID AFTER THIS SESSION

```
SEND 'YOUR ACCT NO. INVALID AFTER THIS SESSION',USER=(T24,OTO)
```

Example 3

Operation: Delete a message.

Known: The message number..... 8

```
SEND 8
```

Example 4

Operation: Have all messages displayed at your terminal.

```
SEND LIST
```

Use the STOPMN subcommand to terminate the monitoring operations of the MONITOR subcommand. This subcommand will halt the display of status information at your terminal.

SUBCOMMAND	OPERANDS
{STOPMN / PM}	{JOBNAMES / SPACE / DSNAME / SESS / STATUS}

JOBNAMES
specifies that the operations provided by the JOBNAMES operand of the MONITOR subcommand are to be stopped. (The system will stop displaying the names of jobs as they start and end.)

SPACE
specifies that the operations provided by the SPACE operand of the MONITOR subcommand are to be stopped. (The system will stop displaying the available space on direct access devices.)

DSNAME
specifies that the operations provided by the DSNAME operand of the MONITOR subcommand are to be stopped. (The system will stop displaying the name of the first non-temporary data set allocated to the volume to which the mount and K type demount messages refer.)

SESS
specifies that the operations provided by the SESS operand of the MONITOR subcommand are to be stopped. (The system will stop notifying the operator whenever a terminal session is initiated or terminated.)

STATUS
specifies that the operations provided by the STATUS operand of the MONITOR subcommand are to be stopped. (The system will stop displaying the names and volume serial numbers of data sets with dispositions of KEEP, CATLG, or UNCATLG at job step end and job end.)

Example 1

Operation: Stop the display of the names of jobs as they begin execution and terminate.

```
STOPMN JOBNAMES
```

Example 2

Operation: Stop the display of available space on direct access devices.

```
stopmn space
```

Use the OUTPUT command to:

- Direct the output from a conventional batch job to your terminal. The output includes the job's Job Control Language statements (JCL), system messages, and system output (SYSOUT) data sets.
- Direct the output from a conventional batch job to a specific data set.
- Change the output class for a conventional batch job.
- Delete the output (SYSOUT) data sets or the system messages for conventional batch jobs.

COMMAND	OPERANDS
{OUTPUT / OUT}	(job-name-list) [CLASS(class-name-list)] [PRINT[*/data-set-name] / NOPRINT[(class-name)]] [NEXT / HERE / BEGIN] [PAUSE / NOPAUSE]

(job-name-list)
specifies one or more names of jobs that have been submitted for conventional batch processing. Each jobname must begin with your user identification (see data set naming conventions) unless the routine that scans and checks the user identification is replaced by a user-written routine. The system will process the output from the jobs identified by the job-name-list.

CLASS(class-name-list)
specifies the names of the output classes to be searched for output from the jobs identified in the jobname list. If you do not specify the name of an output class, the system's default class will be searched for the jobs output. A class name is a single character or digit (A-Z or 0-9). See the publication IBM System/360 Operating System: Supervisor Services and Macro Instructions, GC28-6646, for additional information.

PRINT(data-set-name or *)
is the name of the data set to which the output is to be directed. You may substitute an asterisk for the data set name to indicate that the output is to be directed to your terminal. If you omit both the data set name and the asterisk, the default value is the asterisk. Print is the default value if you omit both PRINT and NOPRINT.

NOPRINT(class-name)
indicates that the output is to be removed from the class specified in the CLASS operand, and placed in the class specified in NOPRINT. If you specify NOPRINT without including a class name, the output is deleted from the system.

Note: Do not specify the following characters as the character in the class-name; the system will try to interpret them as a class-name and thus cause you to lose your data.

comma
tab
blank space
asterisk
semicolon
slash
right parenthesis

NEXT
indicates that output operations of a job that has been interrupted are to be resumed with the next SYSOUT data set or group of system messages.

HERE
indicates that output operations of a job that has been interrupted are to be resumed at a point approximately ten lines before the point of interruption (that is, approximately ten lines will be repeated). This is the default value if you omit HERE, BEGIN, and NEXT.

BEGIN
indicates that output operations of a job that has been interrupted are to be resumed from the beginning of the data set being processed, or from the first message if a block of system messages is being processed.

PAUSE
indicates that output operations are to pause after each SYSOUT data set is listed to allow you to enter a SAVE or CONTINUE subcommand. (A carrier return entered after the pause will cause normal processing to continue.) This operand can be overridden by the NOPAUSE operand of the CONTINUE subcommand.

NOPAUSE
indicates that output operations are not to be interrupted. This operand can be overridden by the PAUSE operand of the CONTINUE subcommand.

Considerations: The OUTPUT command applies to all conventional batch jobs whose job names begin with your user identification. Access to jobs whose job names do not begin with a valid user identification must be provided by a user-written routine. The SUBMIT, STATUS, and CANCEL commands also apply to conventional batch jobs. You must have special permission to use these commands.

Note: You can simplify the use of the OUTPUT command by including the NOTIFY keyword for the SUBMIT command when you submit a job for conventional batch processing. The system will notify you when the job terminates, giving you an opportunity to use the OUTPUT command. SYSOUT data sets should be assigned to SYSOUT classes that do not have conventional output writers operating.

Output Sequence: Output will be produced according to the sequence of the classes that you specify for the CLASS operand. For example, assume that you want to retrieve the output of the following jobs:

```
//JWSD581   JOB   91435,MSGCLASS=X
//           EXEC  PGM=IEBPTPCH
//SYSPRINT  DD    SYSOUT=Y
//SYSUT1    DD    DSNAME=PDS,UNIT=2311,VOL=SER=111112,LABEL=(,SUL),
//                DISP=(OLD,KEEP),DCB=(RECFM=U,BLKSIZE=3625)
//SYSUT2    DD    SYSOUT=Z
//SYSIN     DD    *
          PRINT TYPORG=PS,TOTCONV=XE
          LABELS DATA=NO
/*
//JWSD582   JOB   91435,MSGCLASS=X
//           EXEC  PGM=IEHPROGM
//SYSPRINT  DD    SYSOUT=Y
//DD2       DD    UNIT=2311,VOL=SER=231100,DISP=OLD
//SYSIN     DD    *
//         SCRATCH VTOC,VOL=2311=231100
/*
```

To retrieve the output, you enter:

```
OUTPUT  (JWSD581  JWSD582)   CLASS (X Y Z)
```

Your output will be listed in the following order:

1. Output of class X (JCL and messages for both jobs).

2. Output of class Y (SYSPRINT data for job JWSD581 followed by SYSPRINT data for job JWSD582).

3. Output of class Z (SYSUT2 data for job JWSD581).

Because of this, you should avoid unnecessary division of data sets among classes. If a job uses several classes, you should retrieve the output for that job alone rather than specifying a list of jobnames. By retrieving the job alone, all its output will be together physically.

Subcommands: Subcommands for the OUTPUT command are: CONTINUE, END, and SAVE. When output has been interrupted, you can use the CONTINUE subcommand to resume output operations.

Interruptions occur when:

- Processing of a sysout data set completes and the PAUSE operand was specified with the OUTPUT command.

- Processing of a sysout data set terminates because of an error condition.

- You press the attention key.

- The END subcommand is entered before completion of the job that is being processed.

You can use the SAVE subcommand to rename and catalog a SYSOUT data set for retrieval by a different method. Use the END subcommand to terminate OUTPUT. The remaining portion of a job that has been interrupted will be returned to the output queue.

Example 1

Operation: Direct the output from a conventional batch job to your terminal.

Known: The name of the job..... SMITH2
The job is in the system output class.... SYSOUT=X
Output operations are to be resumed with the next SYSOUT data set or group of system messages.
You want the system to pause after processing each block of output data.

```
OUTPUT SMITH2 CLASS(X) PRINT(*) NEXT PAUSE
```

Example 2

Operation: Direct the output from two conventional batch jobs to a data set so that it can be saved and processed at a later date.

Known: The names of the jobs..... JANA
JANB
The name for the output data set..... JAN.AUGPP.OUTLIST

```
OUTPUT (JANA,JANB) CLASS(R,S,T) PRINT(AUGPP.OUTLIST)
```

Example 3

Operation: Change the output class.

Known: The name of the job..... KEAN1
The existing output class..... SYSOUT=S
The new output class..... T

```
OUTPUT KEAN1 CLASS(S) NOPRINT(T)
```

Example 4

Operation: Delete the output instead of changing the class (see Example 3).

```
OUT KEAN1 CLASS(S) NOPRINT
```

Use the CONTINUE subcommand to resume output operations that have been interrupted.

Interruptions occur when:

- An output operation completes and the PAUSE operand was specified with the OUTPUT command.
- An output operation terminates because of an error condition.
- You press the attention key.

If other TSO commands have been entered during the interruption, the OUTPUT command must be reentered.

SUBCOMMAND	OPERANDS
CONTINUE	[NEXT / HERE / BEGIN] [PAUSE / NOPAUSE]

NEXT
specifies that output operations are to be resumed with the next data set being processed or with the next message if a block of system messages is being processed. This is the default value if NEXT, HERE, and BEGIN are omitted.

HERE
indicates that output operations are to be resumed at a point approximately ten lines before the point of interruption (that is, approximately ten lines will be repeated).

BEGIN
indicates that output operations are to be resumed from the beginning of the data set being processed or from the first message if a block of system messages is being processed.

PAUSE
indicates that output operations are to pause after each data set is processed to allow you to enter a SAVE subcommand. (A carrier return entered after the pause will cause normal processing to continue.) You can use this operand to override a previous NOPAUSE condition for output.

NOPAUSE
indicates that output operations are not to be interrupted. You can use this operand to override a previous PAUSE condition for output.

Example 1

Operation: Continue output operations with the next SYSOUT data set or group of messages.

```
CONTINUE
```

Example 2

Operation: Start output operations over again.

```
CONTINUE BEGIN
```

Use the END subcommand to terminate the operations of the OUTPUT command.

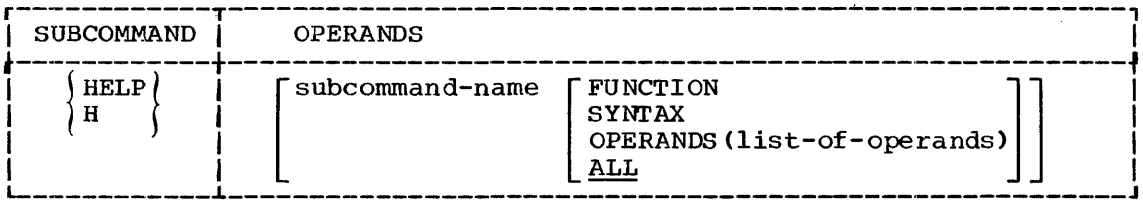
SUBCOMMAND	OPERANDS
END	

HELP Subcommand of OUTPUT

Use the HELP subcommand to find out how to use the OUTPUT subcommands. When you enter the HELP subcommand, the system responds by printing out explanatory information at your terminal. You may request:

- A list of available subcommands.
- An explanation of the function, syntax, and operands of a specific subcommand.

The HELP subcommand actually causes the system to execute a function of the HELP command; therefore, you may consult the discussion of the HELP command if you desire more detailed information.



subcommand-name
specifies the subcommand that you want to have clarified. If you omit this operand, the system will display a list of OUTPUT subcommands.

FUNCTION
specifies that you want a description of the referenced subcommand's function.

SYNTAX
specifies that you want a definition of the proper syntax for the referenced subcommand.

OPERANDS (list-of-operands)
specifies that you want an explanation of the operands applicable to the referenced subcommand.

The list of operands specifies the particular keywords that you want to have explained. If you do not specify any keywords, all of the applicable keywords will be included. You must use one or more blanks or a comma as a delimiter between the keywords in the list.

ALL
specifies that you want a description of the function, the syntax, and the operands of the subcommand that you specified. This is the default value if no operands are specified.

Example 1

Operation: Have a list of available subcommands displayed at your terminal.

```
HELP
```

Example 2

Operation: Obtain all available information about a particular subcommand.

Known: The subcommand name..... SAVE

```
H SAVE
```

Example 3

Operation: Have a list of the operands for a particular subcommand displayed at your terminal.

Known: The subcommand name..... CONTINUE

```
h continue operands
```

SAVE Subcommand of OUTPUT

Use the SAVE subcommand to rename and catalog a SYSOUT data set for retrieval by some method other than the OUTPUT command. To use SAVE, you should have specified the PAUSE keyword on the OUTPUT command.

SUBCOMMAND	OPERANDS
{SAVE / S}	data-set-name

data-set-name
specifies the new data set name to be given to the SYSOUT data set (see the data set naming conventions). The renamed data set will be cataloged by the new name.

Example 1

Operation: Save an output data set.

Known: The name of the data set..... ADT023.NEWOUT.OUTLIST

```
SAVE NEWOUT
```

Use the PROFILE command to establish your user profile; that is, to tell the system how you want to use your terminal. You can:

- Define a character-deletion or line-deletion control character.
- Specify whether or not prompting is to occur.
- Specify whether or not you will accept messages from other terminals.
- Specify whether or not you want the opportunity to obtain additional information about messages from a command procedure.
- Specify whether or not you want message numbers for diagnostic messages that may be displayed at your terminal.

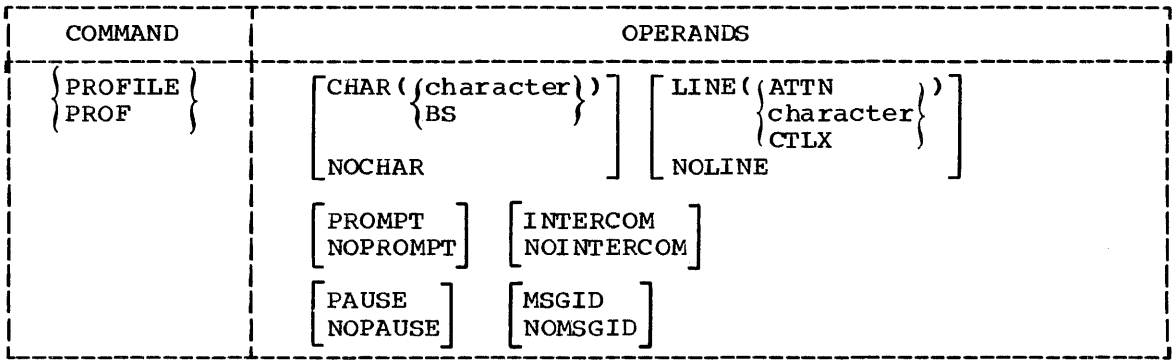
Initially, a user profile is prepared for you when arrangements are made for you to use the system. The authorized system programmer uses the ACCOUNT command to create your userid and your user profile. Under the ACCOUNT command, the system programmer is restricted to defining the same user profile for every userid that he creates. This "typical" user profile is defined when a User Profile Table (UPT) is initialized to hexadecimal zeroes for any new userid. Thus, your initial user profile is made up of the default values of the operands discussed under this command. The system defaults provided for the character-delete and the line-delete control characters depend upon what type of terminal is involved:

TSO Terminal	Character-Delete Control Character	Line-Delete Control Character
IBM 2741 Communication Terminal	BS (backspace)	ATTN (attention)
IBM 1052 Printer-Keyboard	BS (backspace)	**
IBM 2260 Display Station	None	None
IBM 2265 Display Station	None	None
Teletype* Model 33	**	**
Teletype* Model 35	**	**

* Trademark of Teletype Corporation.
** Refer to the publication, IBM System/360 Operating System: Time Sharing Option, Terminals, GC28-6762.

CAUTION: Although highly unlikely, it is possible for the system programmer who created your userid (and therefore your user profile) to have then logged on under it and by using the PROFILE command to have created a unique user profile (different from the "typical" user profile created under the ACCOUNT command) for you. In case of doubt, have the system programmer use the LIST subcommand of ACCOUNT to list your current user profile.

You change the characteristics of your user profile by using the PROFILE command with the appropriate operands. Only the characteristics that you specify explicitly by operands will change; other characteristics remain unchanged. The new characteristics will remain valid from session to session. You must specify at least one operand or the system will ignore the command.



CHAR(character)
specifies the character that you want to use to tell the system to delete the previous character entered. You should not specify a blank, tab, comma, asterisk, or parenthesis because these characters are used to enter commands.

Note: Do not use an alphabetic character as either a character-delete or a line-delete character. For if you do, you run the risk of not being able to enter certain commands without accidentally deleting characters or lines of data. For instance: if you specify R as a character-delete character, each time you tried to enter a PROFILE command the R in PROFILE would delete the P that precedes it. Thus it would be impossible to enter the PROFILE command as long as R was the character-delete control character.

CHAR(BS)
specifies that a backspace signals that the previous character entered should be deleted. This is the default value set when your user profile was created.

NOCHAR
specifies that no control character is to be used for character deletion.

LINE(character)
specifies a control character that you want to use to tell the system to delete the current line.

LINE(ATTN)
specifies that an attention interruption is to be interpreted as a line-deletion control character. This is the default value set when your user profile was created.

LINE(CTLX)
specifies that the X and CTRL keys (depressed together) on a teletype terminal are to be interpreted as a line-deletion control character. This is the default value set when your user profile was created, if you are operating a teletype terminal.

NOLINE
specifies that no line-deletion control character (including ATTN) is recognized.

PROMPT
specifies that you want the system to prompt you for missing information. This is the default value set when your user profile was created.

NOPROMPT
specifies that no prompting is to occur.

INTERCOM
specifies that you are willing to receive messages from other terminal users. This is the default value set when your user profile was created.

NOINTERCOM
specifies that you do not want to receive messages from other terminals.

PAUSE
specifies that you want the opportunity to obtain additional information when a message is issued at your terminal while a command procedure (see the EXEC command) is executing. After a message that has additional levels of information is issued, the system will display the word PAUSE and wait for you to enter a question mark (?) or a carrier return.

NOPAUSE
specifies that you do not want prompting for a question mark or carriage return. This is the default value set when your user profile was created.

MSGID
specifies that diagnostic messages are to include message identifiers.

NOMSGID
specifies that diagnostic messages are not to include message identifiers. This is the default value set when your user profile was created.

Example 1

Operation: Establish a complete user profile

Known: The character that you want to use to tell the system to delete the previous character..... #.
The indicator that you want to use to tell the system to delete the current line..... ATTN.
You want to be prompted.
You do not want to receive messages from other terminals.
You want to be able to get second level messages while a command procedure is executing.
You do not want diagnostic message identifiers.

```
PROFILE CHAR(#) LINE(ATTN) PROMPT NOINTERCOM PAUSE NOMSGID
```

Example 2

Operation: Suppose that you have established the user profile in Example 1. The terminal that you are using now does not have a key to cause an attention interrupt. You want to change the line delete control character from ATTN to @ without changing any other characteristics.

```
PROF LINE(@)
```

Example 3

Operation: Establish and use a line-deletion character and a character-deletion character.

Known: The line-deletion character..... &
The character-deletion character..... !

```
PROFILE LINE(&) CHAR(!)
```

Now, if you type:

```
NOW IS THE TI&AC!BCG!.
```

and press the carrier return key, you will actually enter:

```
ABC.
```

Use the PROTECT command to prevent unauthorized access to your data set. This command establishes or changes:

- The passwords that must be specified to gain access to your data set.
- The type of access allowed.

Data sets that have been allocated (either during a LOGON procedure or via the ALLOCATE command) cannot be protected by specifying the PROTECT command. To password-protect an allocated data set, you would have to de-allocate it via the FREE command before you could protect it via the PROTECT command.

Passwords

You may assign one or more passwords to a data set. Once assigned, the password for a data set must be specified in order to access the data set. A password consists of one through eight alphameric characters. You are allowed two attempts to supply a correct password.

Types of Access

Four operands determine the type of access allowed for your data set. They are, PWREAD, PWWRITE, NOPWREAD, NOWRITE.

Each operand, when used alone, defaults to one of the preceding types of access. The default values for each operand used alone are:

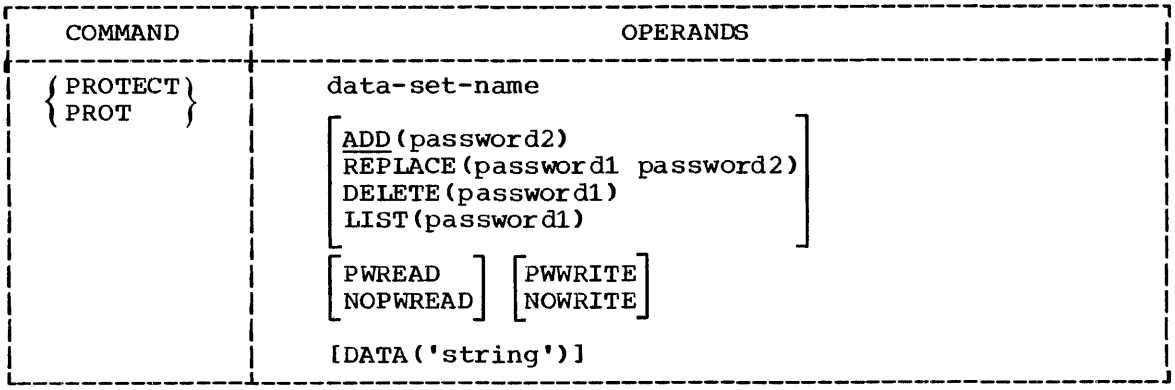
OPERAND	DEFAULT VALUE
PWREAD	PWREAD PWWRITE
NOPWREAD	NOPWREAD PWWRITE
PWWRITE	NOPWREAD PWWRITE
NOWRITE	PWREAD NOWRITE

A combination of NOPWREAD and NOWRITE is not supported and will default to NOPWREAD and PWWRITE.

If you specify a password but do not specify a type of access, the default is:

- NOPWREAD PWWRITE if the data set does not have any existing access restrictions.
- The existing type of access if a type of access has already been established.

When you specify the REPLACE function of the PROTECT command the default type of access is that of the entry being replaced.



data-set-name
specifies the name of the data set that will be subject to the functions of this command (see the data set naming conventions).

ADD(password2)
specifies that a new password is to be required for access to the named data set. This is the default value if ADD, REPLACE, DELETE, and LIST are omitted.

If the data set exists and is not already protected by a password, its security counter will be set and the password being assigned will be flagged as the control password for the data set. The security counter is not affected when additional passwords are entered.

REPLACE(password1, password2)
specifies that you want to replace an existing password, access type, or optional security information. The first value (password1) is the existing password; the second value (password2) is the new password.

DELETE(password1)
specifies that you want to delete an existing password, access type, or optional security information.

If the entry being removed is the control entry (see the discussion following these operand descriptions), all other entries for the data set will also be removed.

LIST(password1)
specifies that you want the security counter, the access type, and any optional security information in the Password Data Set entry to be displayed at your terminal.

password1
specifies the existing password that you want to replace, delete, or have its security information listed.

password2
specifies the new password that you want to add or to replace an existing password.

PWREAD
specifies that the password must be given before the data set can be read.

NOPWREAD
specifies that the data set can be read without using a password.

PWWRITE
specifies that the password must be given before the data set can be written upon.

NOWRITE
specifies that the data set cannot be written upon.

DATA('string')
specifies optional security information to be retained in the system. The value that you supply for 'string' specifies the optional security information that is to be included in the Password Data Set entry (up to 77 bytes).

Password Data Set

Before you can use the PROTECT command, a Password Data Set must reside on the system residence volume. The Password Data Set contains passwords and security information for protected data sets. You can use the PROTECT command to display this information about your data sets at your terminal.

The Password Data Set contains a security counter for each protected data set. This counter keeps a record of the number of times an entry has been referred to. The counter is set to 'zero' at the time an entry is placed into the data set, and is incremented each time the entry is accessed.

Each password is stored as part of an entry in the Password Data Set. The first entry in the Password Data Set for each protected data set is called the control entry. The password from the control entry must be specified for each access of the data set via the PROTECT command, with one exception: the LIST operand of the PROTECT command does not require the password from the control entry.

If you omit a required password when using the PROTECT command, the system will prompt you for it; and if your terminal is equipped with the 'print-inhibit' feature, the system will disengage the printing mechanism at your terminal while you enter the password in response. However, the 'print-inhibit' feature is not used if the prompting is for a new password.

Example 1

Operation: Establish a password for a new data set.

Known: The name of the data set..... LEOBTG.SALES.DATA
The password..... L82GRIFN
The type of access allowed..... PWREAD PWWRITE
The logon id was..... LEOBTG

```
PROTECT SALES.DATA PWREAD ADD(L82GRIFN)
```

Example 2

Operation: Replace an existing password without changing the existing access type.

Known: The name of the data set..... TCOSALES.NETSALES.DATA
The existing password..... MTG@AOP
The new password..... PAO$TMG
The control password..... ELHAVJ
The logon id was..... TCOSALES

```
PROT NETSALES.DATA/ELHAVJ REPLACE(MTG@AOP,PAO$TMG)
```

Example 3

Operation: Delete one of several passwords.

Known: The name of the data set..... MTGGO.NETGROSS.ASM
The password..... LETGO
The control password..... APPLE
The logon id was..... MTGGO

```
      PROT NETGROSS.ASM/APPLE DELETE(LETGO)
```

Example 4

Operation: Obtain a listing of the security information for a protected data set.

Known: The name of the data set..... LTG24.BILLS.CNTRLA
The password required..... D#JPJAM

```
protect 'ltg24.bills.cntrla' list(d#jpjam)
```

Example 5

Operation: Change the type of access allowed for a data set.

Known: The name of the data set..... GJPD23A.PROJCTN.LOAD
The new type of access..... NOPWREAD PWWRITE
The existing password..... DDAY6/6
The control password..... EEYORE
The logon id was..... GJPD23A

```
PROTECT PROJCTN.LOAD/EEYORE REPLACE(DDAY6/6,DDAY6/6)-
NOPWREAD PWWRITE
```

Use the RENAME command to:

- Change the name of a cataloged data set.
- Change the name of a member of a partitioned data set.
- Create an alias for a member of a partitioned data set.

COMMAND	OPERANDS
{RENAME / REN}	old-name new-name [ALIAS]

old-name
: specifies the name that you want to change. The name that you specify may be the name of an existing data set or the name of an existing member of a partitioned data set. (See the data set naming conventions.)

new-name
: specifies the new name to be assigned to the existing data set or member. If you are renaming or assigning an alias to a member, you may supply only the member name and omit all other levels of qualification. (See data set naming conventions).

ALIAS
: specifies that the member name supplied for new name operand is to become an alias for the member identified by the old name operand.

You can rename several data sets by substituting an asterisk for a qualifier in the old name and new name operands. The system will change all data set names that match the old name except for the qualifier corresponding to the asterisk's position.

Example 1

Operation: you have several data sets named:

USERID.MYDATA.DATA

USERID.YOURDATA.DATA

USERID.WORKDATA.DATA

that you want to rename:

USERID.MYDATA.TEXT

USERID.YOURDATA.TEXT

USERID.WORKDATA.TEXT

you must specify either:

```
RENAME 'USERID.*.DATA','USERID.*.TEXT'
```

or

```
RENAME *.DATA,*.TEXT
```

Example 2

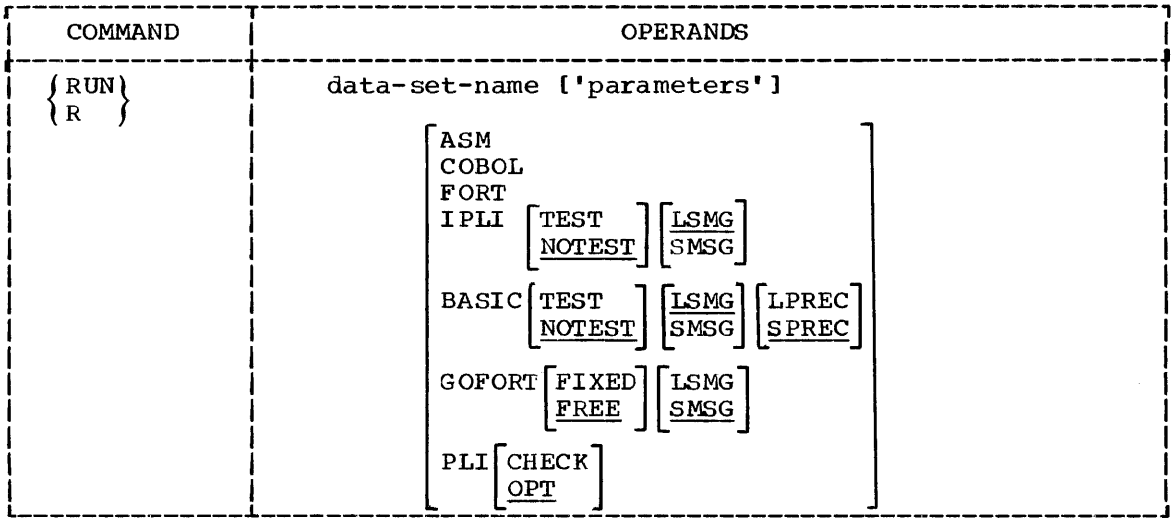
Operation: Assign an alias "SUZIE" to the partitioned data set member named "ELIZBETH(LIZ)".

```
REN  'ELIZBETH(LIZ)' (SUZIE) ALIAS
```

Use the RUN command to compile, load, and execute the source statements in a data set. The RUN command is designed specifically for use with certain program products; it selects and invokes the particular program product needed to process the source statements in the data set that you specify. The following table shows which program product is selected to process each type of source statement. (Appendix A contains references to additional information about the program products.)

If your program or data set contains statements of this type (see EDIT):	Then the following Program Product is needed:
ASM	TSO ASM Prompter
BASIC	ITF:BASIC (Shared Language Component and BASIC Processor)
COBOL	TSO COBOL Prompter and American National Standard COBOL Version 3 Compiler
FORT	TSO FORTRAN Prompter and FORTRAN IV (GI) Compiler
GOFORT	Code and Go FORTRAN
IPLI	ITF:PL/1 (Shared Language Component and PL/1 Processor)
PLI	PL/I Checkout Compiler or PL/I Optimizing Compiler
Programs containing statements suitable for the following IBM-supplied language processors can be compiled and executed by using the CALL command: ASM(F),PL/1(F),FORTRAN(E),(G) or (H) You can use the CONVERT command to convert ITF:PL/I and Code and Go FORTRAN statements to a form suitable for the PL/1 and FORTRAN compilers, respectively.	

The RUN command and the RUN subcommand of EDIT perform the same basic function.



data-set-name 'parameters'
specifies the name of the data set containing the source program. (See the data set naming conventions.) A string of up to 100 characters can be passed to the program via the "parameters" operand (valid only for data sets which accept parameters).

ASM
specifies that the TSO Assembler Prompter Program Product and the Assembler (F) compiler are to be invoked to process the source program. If the rightmost qualifier of the data set name is ASM, this operand is not required.

COBOL
specifies that the TSO COBOL Prompter and the American National Standard COBOL Program Products are to be invoked to process the source program. If the rightmost qualifier of the data set name is COBOL, this operand is not required.

FORT
specifies that the TSO FORTRAN Prompter and the FORTRAN IV (GI) Program Products are to be invoked to process the source program. If the rightmost qualifier of the data set name is FORT, the Code and Go Fortran compiler will be invoked unless you specify this operand.

IPLI
specifies that the ITF:PL/I Program Product is to be invoked to process the source program. If the rightmost qualifier of the data set name is IPLI, this operand is not required.

BASIC
specifies that the ITF:BASIC Program Product is to be invoked to process the source program. If the rightmost qualifier of the data set name is BASIC, this operand is not required.

GOFORT
specifies that the Code and Go Fortran Program Product is to be invoked for interactive processing of the source program.

TEST
specifies that testing of the program is to be performed. This operand is valid only for the ITF:PL/I and BASIC Program Product.

NOTEST
specifies that the TEST function is not desired. This is the default value if both TEST and NOTEST are omitted.

LMSG
specifies that the long form of the diagnostic messages are to be provided. This operand is applicable to the ITF:PL/I, ITF:BASIC, and Code and Go FORTRAN Program Products only. The default value for the LMSG/SMSG operand pair depends on the Program Product being used, as follows:

Program Product	Default Operand
Code and Go	SMSG
ITF:BASIC	LMSG
ITF:PL/I	LMSG

SMSG
specifies that the short form of the diagnostic messages is to be provided. This operand is applicable to the ITF:PL/I, ITF:BASIC, and Code and Go FORTRAN Program Products only.

LPREC
specifies that long precision arithmetic calculations are required by the program. This operand is valid only for the ITF:BASIC Program Product.

SPREC
specifies that short precision arithmetic calculations are adequate for the program. This operand is valid only for the ITF:BASIC Program Product. This is the default value if both LPREC and SPREC are omitted.

FIXED
specifies the format of the source statements to be processed by the Code and Go FORTRAN Program Product. The statements must be in standard format when this operand is specified. If you omit this operand, the FREE operand is the default value.

FREE
specifies that the source program consists of free form statements applicable only to the Code and Go FORTRAN Program Product.

PLI
specifies that the PL/I Program Product is to be invoked to process the source program. If the rightmost qualifier of the data set name is PLI, this operand is not required.

CHECK
specifies the PL/I Checkout Compiler. This operand is valid for the PL/I Program Product only. If you omit this operand, the OPT operand is the default value.

OPT
specifies the PL/I Optimizing Compiler. This operand is valid for the PL/I Program Product only. This is the default value if both CHECK and OPT are omitted.

Determining Compiler Type: The system uses two sources of information to determine which compiler will be used. The first source of information is the optional operand (ASM, COBOL, FORT, IPLI, BASIC, or GOFORT) that you may specify for the RUN command. If you omit this operand, the system checks the descriptive qualifier of the data set name that is to be executed (see the data set naming conventions for a list of descriptive qualifiers). If the system cannot determine the compiler type from the descriptive qualifier, you will be prompted.

Example 1

Operation: Compile, load, and execute a source program composed of BASIC statements.

Known: The name of the data set containing the source program..... DDG39T.MANHRS.BASIC

```
RUN MANHRS.BASIC
```

Example 2

Operation: Compile, load and execute a Code and Go FORTRAN source program contained in a data set that does not conform to the data set naming conventions.

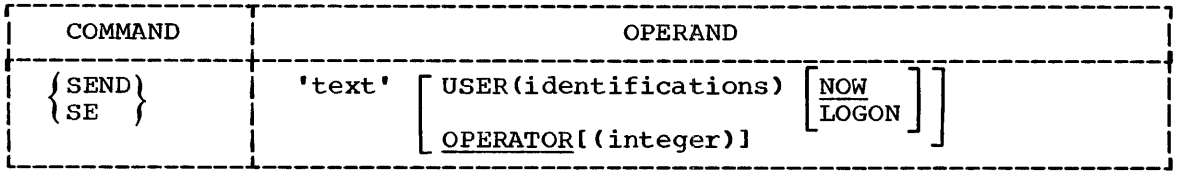
Known: The data set name..... TRAJECT.MISSILE
For FORTRAN statements that conform to the standard format.
Complete diagnostic messages are needed.
Parameters to be passed to the program are... 50 144 5000

```
RUN 'TRAJECT.MISSILE' '50 144 5000' GOFORT FIXED LMSG
```

SEND Command

Use the SEND command to send a message to another terminal user or to the system operator. A message may be sent to more than one terminal user. If the intended recipient of a message is not logged on, the message can be retained within the system and presented automatically when he logs on. You will be notified when the recipient is not logged on and the message is deferred.

This command should be used by terminal users; system operators should use the SEND subcommand of the OPERATOR command.



'text'
specifies the message to be sent. You must enclose the text of the message within apostrophes (single quotes). The message must not exceed 115 characters including blanks. If no other operands are used, the message goes to the console operator. If you want apostrophes to be printed you must enter two in order to get one.

USER(identifications)
specifies the user identification of one or more terminal users who are to receive the message. A maximum of 20 identifications can be used.

NOW
specifies that you want the message to be sent immediately. If the recipient is not logged on, you will be notified and the message will be deleted. This is the default value if both NOW and LOGON are omitted.

LOGON
specifies that you want the message retained in the SYS1.BRODCAST data set if the recipient is not logged on or is not receiving messages. When the recipient logs on, the message will be removed from the data set and directed to his terminal. If the recipient is currently using the system and receiving messages, the message will be sent immediately.

OPERATOR(integer)
specifies that you want the message sent to the operator indicated by the integer. If you omit the integer, the default is two (2); that is, the message goes to the master console operator. This is the default value if both USER (identifications) and OPERATOR are omitted. The integer corresponds to routing codes for the WTO macro as described in the publication, IBM System/360 Operating System: Supervisor Services and Macro Instructions, GC28-6646.

Example 1

Operation: Send a message to the master console operator.

Known: The message:
WHAT IS THE WEEKEND SCHEDULE?

```
SEND 'WHAT IS THE WEEKEND SCHEDULE?'
```

Example 2

Operation: Send a message to two other terminal users.

Known: The message:
ACCOUNT NUMBER 401288 MUST NOT BE USED ANY MORE.
CHANGE TO ACCOUNT NUMBER 530266.
The user identification for the terminal users..... AMCORP6
AMCORP7

```
SEND 'ACCOUNT NUMBER 401288 MUST NOT BE USED ANY-
MORE. CHANGE TO ACCOUNT NUMBER 530266.'-
USER(AMCORP6,AMCORP7) NOW
```

Example 3

Operation: Send a message that is to be delivered to "JONES" when he begins his terminal session or now if he is currently logged on.

Known: The recipient's user identification..... JONES
The message:
IS YOUR VERSION OF THE SIMULATOR READY?

```
SEND 'IS YOUR VERSION OF THE SIMULATOR READY?' USER(JONES) LOGON
```

Use the STATUS command to have the status of conventional batch jobs displayed at your terminal. You can obtain the status of all batch jobs, of several specific batch jobs, or of a single batch job. The information that you receive for each job will tell you whether it is awaiting execution, is currently executing, or has completed execution.

This command may be used only by personnel who have been given the authority to do so by the installation management.

COMMAND	OPERANDS
{STATUS / ST}	[(jobname-list)]

(jobname-list)
specifies the names of the conventional batch jobs that you want to know the status of. If two or more jobs have the same jobname, the system will only display the status of the first one encountered. When more than one jobname is included in the list, the list must be enclosed within parentheses. If you do not specify any jobnames, you will receive the status of all batch jobs in the system whose jobnames begin with your userid.

Example 1

Operation: Have the status of two batch jobs displayed at your terminal.

Known: The jobnames..... ABJ325A2
ABJ325A3

```
STATUS (ABJ325A2,ABJ325A3)
```

SUBMIT Command

Use the SUBMIT command to submit one or more batch jobs for conventional processing. The SUBMIT command allows a foreground (TSO) user to submit a job(s) for interpretation and execution in the background (MVT). Each job(s) submitted must reside in either a sequential, direct-access data set or in a member of a partitioned data set. Either of these data sets can contain one or more jobs that can be executed via a single entry of SUBMIT. Each job(s) must comprise an input job stream (JCL plus data).

Note: If either of the above types of data sets containing 2 or more jobs is submitted for processing, the following applies:

- The SUBMIT command processor will build a job card for the first job in the data set, if necessary, but will not build job cards for any other jobs in the data set. Any job card you supply should have a job name consisting of your userid and an identifying character. For more information on how to submit a background job, refer to the publication: IBM System/360 Operating System: Time Sharing Option, Terminal User's Guide, GC28-6763.
- If the SUBMIT Processor determines that a job cannot execute properly, the remaining job(s) following it in the data set will not be executed.
- Once the SUBMIT Processor submits a job for processing, errors occurring in the execution of that job have no effect on the submission of any remaining job(s) in that data set.
- Once SUBMIT has successfully submitted a job for conventional background processing, the job's JCL will be interpreted by the TSO Background Reader, using the MVT Initiator/Terminator as a standard MVT job.

This command may be used only by personnel who have been given the authority to do so by the installation management.

COMMAND	OPERANDS
{SUBMIT / SUB}	(data-set-list) [NOTIFY / NONOTIFY]

(data-set-list)
specifies one or more data set names or names of members of partitioned data sets (see the data set naming conventions) that define an input stream (JCL plus data). If you specify more than one data set name, enclose them in parentheses.

NOTIFY
specifies that you are to be notified when your job terminates in the background. If you have elected not to receive messages, the message will be placed in the Broadcast data set. You must then enter LISTBC to receive the message. You may obtain this message by issuing LISTBC or LOGON. This is the default value if both NOTIFY and NONOTIFY are omitted.

NONOTIFY
specifies that no message will be placed in the broadcast data set. This operand is only recognized when no job card has been provided with the job that you are processing.

Example 1

Operation: Submit two jobs for conventional batch processing.

Known: The names of the data sets that contain the jobs:

ABTJQ.STRESS.CNTL
ABTJQ.STRAIN.CNTL

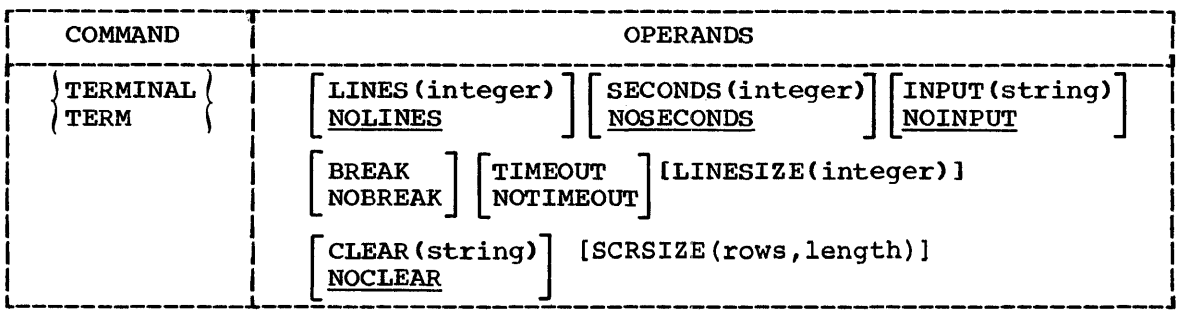
You want to be notified as each job terminates.

```
SUBMIT (STRESS STRAIN)
```

Use the TERMINAL command to define the operating characteristics that depend primarily upon the type of terminal that you are using. You can specify the ways that you want to request an attention interruption and you can identify hardware features and capabilities. The TERMINAL command allows you to request an attention interruption whether or not your terminal has a key for the purpose.

The terminal characteristics that you have defined will remain in effect until you enter the LOGOFF command. If you terminate a session and begin a new one by entering a LOGON command (instead of a LOGOFF command followed by a LOGON command), the terminal characteristics defined in the earlier session will be in effect during the subsequent session.

Refer to IBM System/360 Operating System: Time Sharing Option, Terminals, GC28-6762 for a description of the TERMINAL command's characteristics as they apply to the various terminals available with TSO and for an explanation of how to use the simulated attention facility.



LINES(integer)
specifies an integer from 1 to 255 that indicates you want the opportunity to request an attention interruption after that number of lines of continuous output has been directed to your terminal.

NOLINES
specifies that output line count is not to be used for controlling an attention interruption. This is the default condition.

SECONDS(integer)
specifies an integer from 10 to 2550 (in multiples of 10) to indicate that you want the opportunity to request an attention interruption after that number of seconds has elapsed during which the terminal has been locked and inactive. If you specify an integer that is not a multiple of 10, it will be changed to the next largest multiple of 10.

NOSECONDS
specifies that elapsed time is not to be used for controlling an attention interruption. This is the default condition.

INPUT(string)
specifies the character string that, if entered as input, will cause an attention interruption. The string must be the only input entered and cannot exceed four characters in length.

NOINPUT
specifies that no character string will cause an attention interruption. This is the default condition.

BREAK
specifies that your terminal keyboard will be unlocked to allow you to enter input whenever you are not receiving output from the system; the system can interrupt your input with high-priority messages. Since use of BREAK with a terminal type which cannot support it can result in loss of output or error, check with your installation system manager before specifying this operand.

NOBREAK
specifies that your terminal keyboard will be unlocked only when your program or a command you have used requests input.

Note: The default for the BREAK/NOBREAK operand is determined when your installation defines the terminal features.

TIMEOUT
specifies that your terminal's keyboard will lock up automatically after approximately nine to 18 seconds of no input. (Applicable only to the IBM 1052 Printer-Keyboard without the text timeout suppression feature).

NOTIMEOUT
specifies that your terminal's keyboard will not lockup automatically after approximately nine to 18 seconds of no input. (Applicable only to the IBM 1052 Printer-Keyboard with the text timeout suppression feature.)

Note: The default for the TIMEOUT/NOTIMEOUT operand is determined when your installation defines the terminal features.

LINESIZE(integer)
specifies the length of the line (the number of characters) that can be printed at your terminal. (Not applicable to the IBM 2260 and 2265 Display Stations.) Default values are as follows:

```
IBM 2741 Communication Terminal - 120 characters
IBM 1052 Printer-Keyboard       - 120 characters
Teletype 33/35                  - 72 characters
```

The integer must not exceed 255.

CLEAR(string)
specifies a character string that, if entered as input, will cause the screen of an IBM 2260 or IBM 2265 Display Station to be erased. The 'string' must be the only input entered and cannot exceed four characters in length.

NOCLEAR
specifies that you do not want to use a sequence of characters to erase the screen of an IBM 2260 or IBM 2265 Display Station. This is the default condition.

SCRSIZE(rows,length)
specifies the screen dimensions of an IBM 2260 or IBM 2265 Display Station.
'rows' specifies the maximum number of lines of data that can appear on the screen.

'length' specifies the maximum number of characters in a line of data displayed on the screen.
Valid screen sizes are:

rows,length
6,40
12,40
12,80
15,64

Note: The default values for the SCREEN operand are determined when your installation defines the terminal features.

Example 1

Operation: Modify the characteristics of an IBM 2741 Communication Terminal to allow operation in unlocked-keyboard mode.

Known: Your terminal supports the break facility. The installation has defined a default of NOBREAK for your terminal.

```
TERMINAL BREAK
```

Example 2

Operation: Modify the characteristics of an IBM 1052 Printer-Keyboard whose attention key cannot be used to interrupt output and whose output line size is greater than 80 characters.

Known: You want an opportunity to request an attention interruption after ten consecutive lines of output.
You want to limit the output line length to 80 characters.

```
TERMINAL LINES(10) LINESIZE(80)
```

Example 3

Operation: Establish the characteristics of an IBM 2260 Display Station to allow for attention interruption and screen erasure requests.

Known: You want an opportunity to request an attention interruption if neither input is requested nor output sent for one minute.
You want a $ to stand for an attention interruption request during a regular input operation.
You want a % to stand for a screen erasure request.

```
TERMINAL SECONDS(60) INPUT($) CLEAR(%)
```

TEST Command

Use the TEST command to "debug" a program, that is to test a program for proper execution and to locate any programming errors. To use the TEST command and subcommands, you should be familiar with the basic assembler language and the addressing conventions described in Appendix B. For best results, the program to be tested should be written in basic assembler language. Also, in order to use the symbolic names feature of TEST, the program should have been assembled and link-edited with the TEST operands. For more detail on how to specify the TEST operands, refer to the ASM and/or to the LINK commands in this publication.

Uses of the TEST Command: Before execution begins you can:

- Supply initial values (test data) that you want to pass to the program.
- Establish breakpoints (after instructions) where execution will be interrupted so that you can examine interim results. (Breakpoints should not be inserted into TSO service routines or into any of the TEST load modules.)

You can then execute the program. When you use the TEST command to load and execute a program, the program must be an object module or a load module suitable for processing. If the program that you want to test is already executing, you can begin testing by interrupting the program with an attention interruption followed by the TEST command with no operands. You can also begin testing after an abnormal ending (ABEND) if the program is still in main storage.

Note: If you enter the TEST command without operands, you can test the in-storage copy of your program. If you enter the TEST command with operands, a fresh copy of your program will be brought in for you to test.

Prior to and during execution you can:

- Display the contents of registers and main storage (as when execution is interrupted at a breakpoint).
- Modify the contents of your registers and main storage.
- Display the Program Status Word (PSW).
- List the contents of control blocks.
- "Step through" sections of the program, checking each instruction for proper execution.

Refer to Appendix B for the TEST command addressing conventions.

COMMAND	OPERANDS
TEST	[program-name] ['parameters'] [LOAD / OBJECT] [CP / NOCP]

program-name
specifies the name of the data set containing the program to be tested. (See the data set naming conventions.) The program must be in object module form or load module form.

parameters
specifies a list of parameters to be passed to the named program. The list must not exceed 100 characters including delimiters.

LOAD
specifies that the named program is a load module that has been processed by the linkage editor and is a member of a partitioned data set. This is the default value if both LOAD and OBJECT are omitted.

OBJECT
specifies that the named program is an object module that has not been processed by the linkage editor. The program can be contained in a sequential data set or a member of a partitioned data set.

CP
specifies that the named program is a command processor.

NOCP
specifies that the named program is not a command processor. This is the default value if both CP and NOCP are omitted.

Subcommands: The subcommands of the TEST command are:

ASSIGNMENT OF VALUES( )
modifies values in main storage and in registers.

AT
establishes breakpoints at specified locations.

CALL
initializes registers and initiates processing of the program at a specified address.

COPY
moves data or addresses.

DELETE
deletes a load module.

DROP
removes symbols established by the EQUATE command from the symbol table of the module being tested.

END
terminates all operations of the TEST command and the program being tested.

EQUATE
adds a symbol to the symbol table and assigns attributes and a location to that symbol.

FREEMAIN
frees a specified number of bytes of main storage.

GETMAIN
acquires a specified number of bytes of main storage for use by the program being processed.

GO
restarts the program at the point of interruption or at a specified address.

HELP
lists the subcommands of TEST and explains their function, syntax, and operands.

LIST
displays the contents of main storage area or registers.

LISTDEB
lists the contents of a Data Extent Block (DEB) (you must specify the address of the DEB).

LISTDCB
lists the contents of a Data Control Block (DCB) (you must specify the address of the DCB).

LISTMAP
displays a storage map.

LISTPSW
displays the Program Status Word (PSW).

LISTTCB
lists the contents of the Task Control Block (TCB) (you may specify the address of another TCB).

LOAD
loads a program into main storage for execution.

OFF
removes breakpoints.

QUALIFY
establishes the starting or base location for relative addresses; resolves identical external symbols within a load module.

RUN
terminates TEST and completes execution of the program.

WHERE
displays the absolute address of a symbol or entrypoint or the address of the next executable instruction.

Example 1

Operation: Enter TEST mode after experiencing either an abnormal termination of your program or an interruption.

Known: Either you have received a message saying that your foreground program has terminated abnormally, or, you have struck the attention key while your program was executing. In either case, you would like to begin "debugging" your program.

```
TEST
```

Example 2

Operation: Invoke a program for testing.

Known: The name of the data set that
contains the program..... ABSELF.PAYER.LOAD(THRUST)
The program is a load module and is not a command processor.
The parameters to be passed..... 2048,80

```
TEST PAYER(THRUST) '2048,80'
```

Example 3

Operation: Invoke a program for testing.

Known: The name of the data set that
contains the program..... DECKCO.PAYLOAD.OBJ
The program is an object module and is not a command processor.

```
TEST PAYLOAD OBJECT
```

Example 4

Operation: Test a command processor.

Known: The name of the data set containing
the command processor..... DCOOIL.CMDS.LOAD(OUTPUT)

```
TEST CMDS(OUTPUT) CP
```

When processing is halted at a breakpoint, you can modify values in main storage and in registers. This function is implicit; that is, you do not enter a subcommand name. The system performs the function in response to operands that you enter.

SUBCOMMAND	OPERANDS
	address=data-type'value'

address
specifies the location that you want to contain a new value. The address may be a symbolic address, a relative address, an absolute address, or a register. (See Appendix B for more information about addresses.)

data-type 'value'
specifies the type of data and the value that you want to place in the specified location. You indicate the type of data by one of the following codes:

Code	Type of Data	Maximum Length (Bytes)
C	Character	one line of input[1]
X	Hexadecimal	64
B	Binary	64
H	Fixed point binary (halfword)	6
F	Fixed point binary (fullword)	11
E	Floating point (single precision)	9
D	Floating point (double precision)	18
L	Extended floating point	16
P	Packed decimal	32
Z	Zoned decimal	17
A	Address constant	10
S	Address (base + displacement)	8
Y	Address constant (halfword)	5

[1]Continued lines are permitted.

You include your data following the code. Your data must be enclosed within apostrophes. Any single apostrophes within your data must be coded as two single apostrophes. Character data will be entered as is; all other data types will be translated into upper case (if necessary). A list of data may be specified by enclosing the list in parentheses. The data in the list will be stored in contiguous storage beginning at the location specified by the address operand.

Example 1

Operation: Insert a character string at a particular location in main storage.

Known: The address is a symbol..... INPOINT
The data..... JANUARY 1, 1970

```
INPOINT=C'JANUARY 1, 1970'
```

Example 2

Operation: Insert a binary number into a register.

Known: The number of the register..... Register 6
The data..... 0000 0001 0110 0011

```
6R=B'0000000101100011'
```

Use the AT subcommand to establish breakpoints before the command where processing is to be temporarily halted so that you can examine the results of execution up to the point of interruption.

SUBCOMMAND	OPERANDS
AT	{address[:address] / (address-list)} [(list-of-subcommands)] [COUNT(integer)] [NODEFER / DEFER] [NOTIFY / NONOTIFY]

address
specifies a location that is to contain a breakpoint. The address may be a symbolic address, a relative address, an absolute address, or a general register containing an address. The address must be on a halfword boundary and contain a valid op code. (See Appendix B for more information about addresses.)

address:address
specifies a range of addresses that are to contain breakpoints. Each address may be a symbolic address, a relative address, an absolute address, or a general register containing an address. Each address must be on a halfword boundary. A breakpoint will be established at each instruction between the two addresses. (See Appendix B for more information about addresses.)

address-list
specifies several addresses that are to contain breakpoints. Each address may be a symbolic address, a relative address, an absolute address, or a general register containing an address. The first address must be on a halfword boundary. The list must be enclosed within parentheses, and the addresses in the list must be separated by standard delimiters (one or more blanks or a comma). A breakpoint will be established at each address. (See Appendix B for more information about addresses.)

list-of-subcommands
specifies one or more subcommands to be executed when the program is interrupted at the indicated location. If you specify more than one subcommand, the subcommands must be separated by semicolons (for instance, LISTTCB PRINT(TCBS);LISTPSW;GO CALCULAT). The list cannot be longer than 255 characters.

COUNT(integer)
specifies that processing will not be halted at the breakpoint until it has been encountered a number of times. This operand is directly applicable to program loop situations, where an instruction is executed several times. The breakpoint will be observed each time it has been encountered the number of times specified for the 'integer' operand. The integer specified cannot exceed 32,767.

DEFER
specifies that the breakpoint is to be established in a program that is not yet in storage. The program to contain the breakpoint will be brought in as a result of a LINK, LOAD, ATTACH, or XCTL macro instruction by the program being tested. You must qualify the address of the breakpoint (either LOADNAME.CSECTNAME.RELATIVE or LOADNAME.CSECTNAME.SYMBOL) when you specify this operand. All breakpoint addresses listed in an AT subcommand with the DEFER operand must refer to the same load module.

NODEFER
specifies that the breakpoint is to be inserted into the program now in main storage. This is the default value if both DEFER and NODEFER are omitted.

NOTIFY
specifies that when it is encountered the breakpoint will be identified at the terminal. This is the default value if both NOTIFY and NONOTIFY are omitted.

NONOTIFY
specifies that when it is encountered the breakpoint will not be identified at the terminal.

Example 1

Operation: Establish breakpoints at each instruction in a section of the program that is being tested.

Known: The addresses of the first and last instructions of that section that is to be tested..... LOOPA
EXITA
The subcommands to be executed are..... LISTPSW,GO

```
AT LOOPA:EXITA (LISTPSW;GO)
```

Example 2

Operation: Establish breakpoints at several locations in a program.

Known: The addresses for the breakpoints..... +8A
LOOPB
EXITB

```
AT (+8A LOOPB EXITB)
```

Example 3

Operation: Establish a breakpoint at a location in a loop. The address of the location is contained in register 15. You only want to have an interruption every tenth cycle through the loop.

Known: The address for the breakpoint..... 15R%

```
AT 15R% COUNT(10)
```

Example 4

Operation: Establish a breakpoint for a program other than the one presently in main storage.

Known: The csect name..... YLREVEB
The name of the load module..... KCIW
The symbolic address for the breakpoint..... PROG

```
AT KCIW.YLREVEB.PROG DEFER
```

Use the CALL subcommand to initiate processing at a specified address. You can pass parameters to the program that is to be tested.

CAUTION: The contents of registers 1, 14, and 15 are changed by the use of the CALL subcommand. To save the contents of these registers, use the COPY subcommand of TEST (see examples 2 and 3 under the COPY subcommand).

SUBCOMMAND	OPERANDS
CALL	address [PARM(address-list)] [VL] [RETURN(address)]

address
specifies the address where processing is to begin. The address may be a symbolic address, a relative address, an absolute address, or a register containing an address. Register 15 contains this address when the program under test begins execution. (See Appendix B for more information about addresses.)

PARM(address-list)
specifies one or more addresses that point to data to be used by the program being tested. The list of addresses will be expanded to fullwords and placed into contiguous storage. Register 1 will contain the address of the start of the list. If PARM is omitted, register 1 will point to a fullword that contains the address of a halfword of zeroes.

VL
specifies that the high order bit of the last fullword of the list of addresses pointed to by general register one is to be set to one.

RETURN(address)
specifies that register 14 is to contain the address that you supply as the value for this keyword. After the program executes, the system will return control to the point indicated by register 14. If RETURN is omitted, register 14 will contain the address of a breakpoint instruction.

Example 1

Operation: Initiate execution of the program being tested at a particular location.

Known: The starting address..... +0A
The addresses of data to be passed..... CTCOUNTR
LOOPCNT
TAX

```
CALL +0A PARM(CTCOUNTR LOOPCNT TAX)
```

Example 2

Operation: Initiate execution at a particular location.

Known: The starting address..... STARTBD
The addresses of data to be passed..... BDFLAGS
PRFTTBL
COSTTBL
ERREXIT
Set the high order bit of the last address parameter to one so that the program can tell the end of the list.
After execution, control is to be returned to..... +24A

```
CALL STARTBD PARM(BDFLAGS PRFTTBL COSTTBL ERREXIT)-
VL RETURN(+24A)
```

COPY Subcommand of TEST

Use the COPY subcommand to transfer data or addresses from one main storage address to another, from one general register to another, from a register to main storage, or from main storage to a register.

The COPY subcommand can be used to:
- Save or restore the contents of the general registers.
- Propagate the value of a byte throughout a field.
- Move an entire data field from one location to another.

SUBCOMMAND	OPERANDS
COPY C	address1 address2 [LENGTH integer] 4	POINTER NOPOINT

address1
specifies a location that contains data to be copied. The address may be a symbolic address, a relative address, an absolute address, an indirect address, or a qualified address. (See Appendix B for more information about address types.)

address2
specifies a location that will receive the data after it is copied. The address may be a symbolic address, a relative address, an absolute address, an indirect address, or a qualified address. (See Appendix B for more information about address types.)

LENGTH(integer)
specifies the length, in decimal, of the field to be copied. If an integer is not specified, LENGTH will default to 4 bytes. The LENGTH keyword can also be entered in the shorter form, L(integer).

POINTER
specifies that address1 will be validity checked to see that it does not exceed maximum core size and will then be treated as an immediate operand (hexadecimal literal) with a maximum length of 4 bytes (that is, an address will be converted to its hexadecimal equivalent) and will be transferred into the location specified by address2. When using the POINTER keyword, do not specify a general register as address1. The POINTER keyword can also be entered in the shorter form, P.

NOPOINT
specifies that address1 will be treated as an address. NOPOINT is the default for POINTER.

Note:
1. The COPY subcommand treats the 16 general registers (R0-R15) as contiguous fields, that is, if you have specified that 8 bytes be moved from R0 to another location for example, COPY R0 80060. LENGTH(8), the COPY subcommand will move the 4 bytes of register 0 and the 4 bytes of register 1 to main storage beginning at location 80060. When a register is specified as address1, the maximum length of data that will be transferred is the total length of the general registers, or 64 bytes.
2. When the value of address2 is one greater than address1, propagation of the data in address1 will occur; when the value of address2 is more than one greater than the value of address1, no propagation will occur.

Example 1

Operation: Transfer 2 full words of data from one main storage location to another.

Known: The starting address of the data..... 80680
The starting address of where the data is to be..... 80685

```
COPY 80680. 80685. LENGTH(8)
```

Example 2

Operation: Copy the contents of one register into another register.

Known: The register which contains the data to be copied..... 10
The register which will contain the data..... 5

```
COPY 10R 5R
```

Example 3

Operation: Save the contents of the general registers.

Known: The first register to be saved..... 0
The starting address of the save area..... A0200

```
C 0R A0200. L(64)
```

Example 4

Operation: Propagate the value in the first byte of a buffer throughout the buffer.

Known: The starting address of the buffer..... 80680
The length of the buffer..... 80 bytes

```
C 80680. 80681. L(79)
```

Example 5

Operation: Insert a hexadecimal value into the high-order byte of a register.

Known: The desired value..... X'80'
The register..... 1

```
COPY 80. 1R L(1) POINTER
```

Example 6

Operation: Insert the entry point of a routine into a storage location.

Known: The module name and the entry point name..... IEFBR14.IEFBR14
The desired storage location..... B0200

```
C IEFBR14.IEFBR14 B0200 P
```

Example 7

Operation: Copy the contents of an area pointed to by a register into another area.

Known: The register which points to the area that contains the data to be moved..... 14
The main storage location which is to contain the data.. 80680
The length of the data to be moved..... 8 bytes

```
C 14R% 80680. L(8) NOPOINT
```

Use the DELETE subcommand to delete a load module awaiting execution.

SUBCOMMAND	OPERAND
{DELETE / D}	load-name

load name
specifies the name of the load module to be deleted. The load name is the name by which the program is known to the system when it is in main storage. The name must not exceed eight characters.

Example 1

Operation: The program being tested has called a subroutine that is in load module form. Before executing the subroutine, a breakpoint is encountered. You do not want to execute the subroutine because you intend to pass test data to the program instead. You now want to delete the subroutine since it will not be used.

Known: The name of the subroutine (load module)..... TOTAL

```
DELETE TOTAL
```

or

```
D TOTAL
```

Use the DROP subcommand to remove symbols from the symbol table of the module being tested. You can only remove symbols that you established with the EQUATE subcommand; you cannot remove symbols that were established by the linkage editor.

SUBCOMMAND	OPERAND
DROP	(symbol-list)

(symbol-list)
specifies one or more symbols that you want to remove from the symbol table created by the EQUATE subcommand. When you specify only one symbol, you do not have to enclose that symbol within parentheses; however, if you specify more than one symbol you must enclose them within parentheses. If you do not specify any symbols, the entire table of symbols will be removed.

Example 1

Operation: Remove all symbols that you have established with the EQUATE command.

```
DROP
```

Example 2

Operation: Remove several symbols from the symbol table.

Known: The names of the symbols..... STARTADD
TOTAL
WRITESUM

```
DROP (STARTADD TOTAL WRITESUM)
```

Use the END subcommand to terminate all functions of the TEST command and the program being tested.

SUBCOMMAND	OPERANDS
END	

EQUATE Subcommand of TEST

Use the EQUATE subcommand to add a symbol to the symbol table of the module being tested. This subcommand allows you to establish a new symbol that you can use to refer to an address or to override an existing symbol to reflect a new address or new attributes. If no symbol table exists, one is created and the specified name is added to it. You can also modify the data attributes (type, length, and multiplicity). The DROP subcommand removes symbols added by the EQUATE subcommand. Symbols established via the EQUATE command are defined for the duration of the TEST session, only.

SUBCOMMAND	OPERANDS
{EQUATE / EQ}	symbol address data-type [LENGTH(integer)] [MULTIPLE(integer)]

symbol
: specifies the symbol (name) that you want to have added to the symbol table so that you can refer to an address symbolically. The symbol must consist of one through eight alphameric characters, the first of which is an alphabetic character.

address
: specifies a symbolic address, a relative address, an absolute address, or a general register containing an address. The address that you specify will be equated to the symbol that you specify. (See Appendix B for more information about addresses.)

data-type
: specifies either the type of data that you want moved into the location specified via the "address" operand, or the characteristics you wish to attribute to the data at the location given by "address." These may or may not be the same as the original characteristics. You indicate the type of data by one of the following codes:

Code	Type of Data	Maximum Length (Bytes)
C	Character	256
X	Hexadecimal	256
B	Binary	256
I	Assembler instruction	256
H	Fixed point binary (halfword)	8
F	Fixed point binary (fullword)	8
E	Floating point (single precision)	8
D	Floating point (double precision)	8
L	Extended floating point	16
P	Packed decimal	16
Z	Zoned decimal	16
A	Address constant	4
S	Address (base + displacement)	2
Y	Address constant (halfword)	2

LENGTH(integer)
specifies the length of the data. The maximum value of the integer is 256. If you do not specify the length, the following default values will apply:

Type of Data	Default Length (Bytes)
C,B,P,Z	1
H,S,Y	2
F,E,A,X	4
D	8
I	variable
L	16

MULTIPLE(integer)
specifies a multiplicity factor. The multiplicity factor means that one element of the data appears several times in succession; the number of repetitions is indicated by the number specified for "integer". The maximum value of the integer is 256.

Note: If you do not specify any keywords, the defaults are:

type - X
multiplicity - 1
length - 4

Example 1

Operation: Add a symbolic address to the symbol table of the module that you are testing.

Known: The symbol..... EXITRTN
The address..... TOTAL+4

```
EQUATE EXITRTN TOTAL+4
```

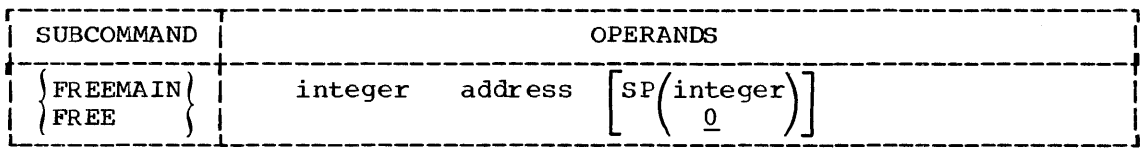
Example 2

Operation: Change the address and attributes for an existing symbol.

Known: The symbol..... CONSTANT
The new address..... 1FAA0.
The new attributes: type..... C
length..... L(8)
multiplicity..... M(2)

```
EQ CONSTANT 1FAA0. C M(2) L(8)
```

Use the FREEMAIN subcommand to free a specified number of bytes of main storage.



integer
specifies the number of bytes of main storage to be released.

address
specifies a symbolic address, a relative address, an absolute address, or a general register containing an address. This address is the location of the space to be freed and must be a multiple of 8 bytes. (See Appendix B for more information about address.)

The LISTMAP subcommand may be used to help locate previously acquired main storage.

SP(integer)
specifies the number of the subpool that contains the space to be freed. If you omit this operand, the default value is subpool zero. The integer must be in the range zero through 127.

Example 1

Operation: Free space in main storage that was acquired previously by a GETMAIN subcommand or by a GETMAIN macro instruction in the module being tested.

Known: The size of the space, in bytes..... 500
The absolute address of the space..... 054A20
The number of the subpool that the space was acquired from..... 3

```
FREE  500  054A20.  SP(3)
```

Use the GETMAIN subcommand to obtain a specified number of bytes of main storage.

SUBCOMMAND	OPERANDS
{GETMAIN / GET}	integer [SP(integer / 0)] [EQUATE(name)]

EQUATE(name)
specifies that the address of acquired storage is to be equated to the symbol specified by "name".

integer
specifies the number of bytes of main storage to be obtained.

SP(integer)
specifies the number of a subpool that contains the bytes of main storage that you want to obtain. If you omit this operand, the default value is subpool zero. The integer must be in the range zero through 127.

Example 1

Operation: Get 500 bytes of main storage from subpool 3 and equate starting address to symbolic name AREA.

```
GET  500  SP(3)  EQUATE(AREA)
```

GO Subcommand of TEST

Use the GO subcommand to start or restart program execution from a particular address. If the program was interrupted for a breakpoint and you want to continue from the breakpoint, there is no need to specify the address. However, you may start execution at any point by specifying the address.

SUBCOMMAND	OPERANDS
GO	[address]

address
: specifies a symbolic address, a relative address, an absolute address, or a general register containing an address. Execution will begin at the address that you specify. (See Appendix B for more information about addresses.)

Example 1

Operation: Begin execution of a program at the point where the last interruption occurred.

```
GO
```

Example 2

Operation: Begin execution at a particular address.

Known: The address..... CALCULAT

```
GO CALCULAT
```

Use the HELP subcommand to find out how to use the TEST subcommands. When you enter the HELP subcommand, the system responds by printing out explanatory information at your terminal. You may request:

- A list of available subcommands.
- An explanation of the function, syntax, and operands of a specific subcommand.

The HELP subcommand actually causes the system to execute a function of the HELP command; therefore, you may consult the discussion of the HELP command if you desire more detailed information.

SUBCOMMAND	OPERANDS
{HELP / H}	[subcommand-name [FUNCTION / SYNTAX / [OPERANDS[(list-of-operands)]] / [ALL]]]

subcommand-name
specifies the subcommand that you want to have clarified. If you omit this operand, the system will display a list of TEST subcommands.

FUNCTION
specifies that you want a description of the referenced subcommand's function.

SYNTAX
specifies that you want a definition of the proper syntax for the referenced subcommand.

OPERANDS(list-of-operands)
specifies that you want an explanation of the operands applicable to the referenced subcommand.

The list of operands specifies the particular keywords that you want to have explained. If you do not specify any keywords, all of the applicable keywords will be included. You must use one or more blanks or a comma as a delimiter between the keywords in the list.

ALL
specifies that you want a description of the function, the syntax, and the operands of the subcommand that you specified. This is the default value if no operands are specified.

Example 1

Operation: Have a list of available subcommands displayed at your terminal.

```
HELP
```

Example 2

Operation: Obtain all available information about a particular subcommand.

Known: The subcommand name..... QUALIFY

```
H QUALIFY
```

Example 3

Operation: Have a list of the operands for a particular subcommand displayed at your terminal.

Known: The subcommand name..... LIST

```
h list operands
```

Use the LIST subcommand to have the contents of a specified area of main storage, or the contents of registers, displayed at your terminal or placed into a data set.

SUBCOMMAND	OPERANDS		
{LIST / L}	{address[:address] / (address-list)}	data-type	[LENGTH(integer)] [MULTIPLE(integer)] [PRINT(data-set-name)]

address
specifies the location of data that you want displayed at your terminal or placed into a data set. The address may be a symbolic address, a relative address, an absolute address, or a general or floating-point register. (See Appendix B for more information about addresses.)

address:address
specifies that you want the data located between the specified addresses displayed at your terminal or placed into a data set. Each address may be a symbolic address, a relative address, an absolute address, or a general or floating point register. (See Appendix B for more information about addresses.)

(address-list)
specifies several addresses of data that you want displayed at your terminal or placed into a data set. The data at each location will be retrieved. Each address may be a symbolic address, a relative address, an absolute address, or a general or floating-point register. The list of addresses must be enclosed within parentheses, and the addresses must be separated by standard delimiters (one or more blanks or a comma). (See Appendix B for more information about addresses.)

data-type
specifies the type of data that is in the specified location. You indicate the type of data by one of the following codes:

Code	Type of Data	Maximum Length (Bytes)
C	Character	256
X	Hexadecimal	256
B	Binary	256
I	Assembler instruction	256
H	Fixed point binary (halfword)	8
F	Fixed point binary (fullword)	8
E	Floating point (single precision)	8
D	Floating point (double precision)	8
L	Extended floating point	16
P	Packed decimal	16
Z	Zoned decimal	16
A	Address constant	4
S	Address (base + displacement)	2
Y	Address constant (halfword)	2

LENGTH(integer)
indicates the length, in bytes of the data that is to be listed. The maximum value for the integer is 256. If you use a symbolic address and do not specify length, the value for the length parameter will be retrieved from the symbol table residing in the user's region. Otherwise, the following default values will apply:

Type of Data	Default Length (Bytes)
C,B,P,Z	1
H,S,Y	2
F,E,A,X	4
D	8
I	variable
L	16

When the data type is I, either length or multiple may be specified, but not both. If both are specified, the multiple parameter is ignored but no error message is printed.

MULTIPLE(integer)
Used in conjunction with the length operand. Gives the user the following options:

- The ability to format the data to be listed (see example 3, below)
- A way of printing more than 256 bytes at a time. (The value supplied for "integer" determines how many "lengths" or multiples of data-type the user wants listed.) The value supplied for integer cannot exceed 256.

For I type data, the value supplied for MULTIPLE defines the number of instructions to be listed. If you use a symbolic address and do not specify MULTIPLE, the value for the MULTIPLE parameter will be retrieved from the symbol table residing in the user's region.

PRINT(data-set-name)
specifies the name of a sequential data set to which the data is directed (see data set naming conventions). If you omit this operand, the data will go to your terminal.

The data format is blocked variable length records. Old data sets with the standard record format and block size are treated as NEW if being opened for the first time, otherwise, they are treated as MOD data sets.

The LIST subcommands of TEST perform the following functions on each data set they process.

If your record format was:	Fixed, Fixed Blocked, or Undefined		Variable or Variable Blocked	
Then it is changed to variable blocked with the following attributes	Recordsize	Blocksize	Recordsize	Blocksize
	125	1629	125	129

Note: Record and block sizes greater than above will be unchanged.

The specified data set is kept open until:

1. The TEST session is ended by a RUN or END subcommand or
2. A LIST subcommand is entered specifying a different PRINT data set. In this case, the previous data set is closed and the current one opened.

Example 1

Operation: List the contents of an area of main storage.

Known:	The area to be displayed is between.....	COUNTERA DTABLE
	The attributes of the data.....	C L(130) M(1)
	The name for a data set to contain the listed data.....	DCDUMP

```
LIST COUNTERA:DTABLE C L(130) M(1) PRINT(DCDUMP)
```

Example 2

Operation: List the contents of main storage at several addresses

Known:	The addresses.....	TOTAL1 TOTAL2 TOTAL3 ALLTOTAL
	The attributes of the data.....	F L(3) M(3)

```
L (TOTAL1 TOTAL2 TOTAL3 ALLTOTAL) F L(3) M(3)
```

Example 3

Operation: List the first six fullwords in the Communications Vector Table (CVT).

Known: The absolute address of the CVT..... 10.
The user is operating in TEST mode.
The attributes of the data..... X
L(4)
M(6)

Note: First use the QUALIFY subcommand of TEST to establish the beginning of the CVT as a base location for displacement values.

```
QUALIFY 10.
```

TEST..... The system response

```
LIST +0 L(4) M(6)
```

The listing at your terminal will resemble the following sample listing:

```
+0  00000000
+4  00012A34
+8  00000B2C
+C  00000000
+10 001A0408
+14 00004430
```

Use the LISTDCB subcommand to list the contents of a data control block (DCB). You must provide the address of the beginning of the DCB. The forty-nine or fifty-two bytes of data following the address will be formatted according to the names of the fields as presented in the publication System/360 Operating System: System Control Blocks, GC28-6628.

If you wish, you can have only selected fields displayed. The field identification is based on the sequential access method DCB for direct access. Fifty-two bytes of data are displayed if the data set is closed; forty-nine bytes of data are displayed if the data set is opened.

SUBCOMMAND	OPERANDS
LISTDCB	address [FIELD(names)] [PRINT(data-set-name)]

address
specifies a symbolic address, a relative address, an absolute address, or a general register containing an address. The specified address is the address of the DCB that you want displayed. The address must be on a fullword boundary. (See Appendix B for more information about addresses.)

FIELD(names)
specifies one or more names of the particular fields in the DCB that you want to have displayed at your terminal. The segment name will not be printed when you use this operand. If you omit this operand, the entire DCB will be displayed.

PRINT(data-set-name)
specifies the name of the sequential data set to which data is to be directed (see data set naming conventions). If you omit this operand, the data will be displayed at your terminal.

The data format is blocked variable length records. Old data sets with the standard record format and blocksize are treated as NEW if they are being opened for the first time; otherwise they are treated as MOD data sets.

The specified data set is kept open until:

1. The TEST session is ended by a RUN or END subcommand, or
2. A LIST subcommand is entered that specifies a different PRINT data set. In this case, the former data set is closed and the current one opened.

Example 1

Operation: List the RECFM field of a DCB for the program that is being tested.

Known: The DCB begins at location..... DCBIN

```
LISTDCB DCBIN FIELD(DCBRECFM)
```

Example 2

Operation: List on entire DCB.

Known: The absolute address of the DCB..... 33B4

```
LISTDCB 33B4.
```

Use the LISTDEB subcommand to list the contents of a data extent block (DEB). You must provide the address of the DEB. The 32 bytes of data following the address will be formatted according to the names of the fields as presented in the publication System/360 Operating System: System Control Blocks, GC28-6628.

In addition to the 32 byte basic section, you may receive up to 16 direct access device dependent sections of 16 bytes each until the full length has been displayed. If you wish, you can have only selected fields displayed.

SUBCOMMAND	OPERANDS
LISTDEB	address [FIELD(names)] [PRINT(data-set-name)]

address
specifies a symbolic address, a relative address, an absolute address, or a general register containing an address. The address is the beginning of the DEB, and must be on a fullword boundary. (See Appendix B for more information about addresses.)

FIELD(names)
specifies one or more names of the particular fields in the DEB that you want to have displayed at your terminal. If you omit this operand, the entire DEB will be listed.

PRINT(data-set-name)
specifies the name of the sequential data set to which data is to be directed (see data set naming conventions). If you omit this operand, the data will be displayed at your terminal.

The data format is blocked variable length records. Old data sets with the standard record format and blocksize are treated as NEW if they are being opened for the first time; otherwise they are treated as MOD data sets.

The specified data set is kept open until:

1. The TEST session is ended by a RUN or END subcommand, or
2. A LIST subcommand is entered that specifies a different PRINT data set. In this case, the former data set is closed and the current one opened.

Example 1

Operation: List the entire DEB for the DCB that is named DCBIN.

Known: The address of the DEB..... DCBIN+2C%

```
LISTDEB DCBIN+2C%
```

Use the LISTMAP subcommand to display a storage map at your terminal. The map identifies the location and assignment of any storage assigned to the program.

All storage assigned to the problem program and its subtasks as a result of GETMAIN requests is located and identified by subpool (0-127). All programs assigned to the problem program and its subtasks are identified by name, size, location, and attribute. Storage assignment and program assignment are displayed by task. When the assignments for the problem program and all its subtasks tasks have been displayed, a map of all unassigned storage within the region is displayed.

SUBCOMMAND	OPERANDS
LISTMAP	[PRINT(data-set-name)]

PRINT(data-set-name)
specifies the name of the sequential data set to which data is to be directed (see data set naming conventions). If you omit this operand, the data will be displayed at your terminal.

The data format is blocked variable length records. Old data sets with the standard record format and blocksize are treated as NEW if they are being opened for the first item; otherwise, they are treated as MOD data sets.

The specified data set is kept open until:

1. The TEST session is ended by a RUN or END subcommand, or
2. A LIST subcommand is entered that specifies a different PRINT data set. In this case, the former data set is closed and the current one opened.

Example 1

Operation: Display a map of main storage at your terminal.

```
LISTMAP
```

Example 2

Operation: Direct a map of main storage to a data set.

Known: The name for the data set..... ACDQP.MAP.TESTLIST

```
LISTMAP PRINT(MAP)
```

Use the LISTPSW subcommand to display a Program Status Word (PSW) at your terminal.

SUBCOMMAND	OPERANDS
LISTPSW	[ADDR(address)] [PRINT(data-set-name)]

ADDR(address)
specifies a symbolic address, a relative address, an absolute address, or a general register containing an address. The address identifies a particular PSW. If you do not specify an address, you will receive the current PSW for the program that is executing. (See Appendix B for more information about addresses.)

PRINT(data-set-name)
specifies the name of the sequential data set to which data is to be directed (see data set naming conventions). If you omit this operand, the data will be displayed at your terminal.

The data format is blocked variable length records. Old data sets with the standard record format and blocksize are treated as NEW if they are being opened for the first time; otherwise, they are treated as MOD data sets.

The specified data set is kept open until:

1. The TEST session is ended by a RUN or END subcommand, or
2. A LIST subcommand is entered that specifies a different PRINT data set. In this case, the former data set is closed and the current one opened.

Example 1

Operation: Display the current PSW at your terminal.

```
LISTPSW
```

Example 2

Operation: Copy the Input/Output old PSW onto a data set.

Known: The address of the PSW (in hexadecimal)..... 38.
The name for the data set..... SKJ23.PSWS.TESTLIST

```
LISTPSW ADDR(38.) PRINT(PSWS)
```

LISTTCB Subcommand of TEST

Use the LISTTCB subcommand to display the contents of a task control block (TCB). You may provide the address of the beginning of the TCB. The data following the address will be formatted according to the names of the fields as presented in the publication: System/360 Operating System: System Control Blocks, GC28-6628.

If you wish, you can have only selected fields displayed.

SUBCOMMAND	OPERANDS
LISTTCB	[ADDR(address)] [FIELD(names)] [PRINT(data-set-name)]

ADDR(address)
specifies a symbolic address, a relative address, an absolute address, or a general register containing an address. The address must be on a fullword boundary. The address identifies the particular TCB that you want to display. If you omit an address, the TCB for the current task is displayed. (See Appendix B for more information about addresses.)

FIELD(names)
specifies one or more names of the particular fields in the TCB that you want to have displayed. If you omit this operand, the entire TCB will be displayed.

PRINT(data-set-name)
specifies the name of the sequential data set to which data is to be directed. If you omit this operand, the data will be displayed at your terminal.

The data format is blocked variable length records. Old data sets with the standard record format and blocksize are treated as NEW if they are being opened for the first time; otherwise, they are treated as MOD data sets.

The specified data set is kept open until:

1. The TEST session is ended by a RUN or END subcommand, or
2. A LIST subcommand is entered that specifies a different PRINT data set. In this case, the former data set is closed and the current one opened.

Example 1

Operation: Save a copy of the TCB for the current task on a data set.

Known: The name for the data set..... GCAMP.TCBS.TESTLIST

```
LISTTCB PRINT(TCBS)
```

Example 2

Operation: Save a copy of some fields of a task's control block that is not active in a data set for future information.

Known: The symbolic address of the TCB..... MYTCB2
The fields that are being requested..... TCBTIO
TCBCMP
TCBGRS
The name for the data set..... SCOTT.TESTLIST

```
LISTTCB ADDR(MYTCB2) FIELD(TCBTIO,TCBCMP,TCBGRS)-
PRINT('SCOTT.TESTLIST')
```

Use the LOAD subcommand to load a program into main storage for execution.

SUBCOMMAND	OPERANDS
LOAD	program-name

program name
specifies the name of a member of a partitioned data set that contains the load module to be tested. (See the data set naming conventions.)

Example 1

Operation: Load a program named ATX03.LOAD(GSCORES)

```
LOAD  (GSCORES)
```

Use the OFF subcommand to remove breakpoints from a program.

SUBCOMMAND	OPERAND
OFF	[address[:address] (address-list)]

address
specifies the location of a breakpoint that you want to remove. The address may be a symbolic address, a relative address, an absolute address, or a general register containing an address. The address must be on a halfword boundary. (See Appendix B for more information about addresses.)

address:address
specifies a range of addresses. Each address may be a symbolic address, a relative address, an absolute address, or a general register containing an address. Each address must be on a halfword boundary. All breakpoints in the range of addresses will be removed. (See Appendix B for more information about addresses.)

(address-list)
specifies the location of several breakpoints that you want to remove. Each address may be a symbolic address, a relative address, an absolute address, or a general register containing an address. Each address must be on a halfword boundary. (See Appendix B for more information about addresses.)

Example 1

Operation: Remove all breakpoints in a section of the program.

Known: The beginning and ending addresses of the section..... LOOPC
EXITC

```
OFF  LOOPC:EXITC
```

Example 2

Operation: Remove several breakpoints located at different positions.

Known: The addresses of the breakpoints..... COUNTRA
COUNTRB
EXITA

```
OFF  (COUNTRA  COUNTRB EXITA)
```

Example 3

Operation: Remove all breakpoints in a program.

```
OFF
```

Use the QUALIFY subcommand to qualify symbolic and relative addresses; that is, to establish the starting or base location to which displacements are added so that an absolute address is obtained. The QUALIFY subcommand allows you to specify uniquely which program and which CSECT within that program you intend to test using symbolic and relative addresses.

You can specify an address to be used as the base location for subsequent relative addresses. Each time you use the QUALIFY subcommand, previous qualifications are voided.

Symbols that were established by the EQUATE subcommand before you enter QUALIFY are not affected by the QUALIFY subcommand.

SUBCOMMAND	OPERANDS
{QUALIFY / Q}	{address / load-module-name[.entryname] [TCB(address)]}

address
specifies an absolute, relative or symbolic address.

load
specifies the name by which a load module is known. The load name may be a member name of a partitioned data set or an alias.

load.entry
specifies the name by which a load module is known, and an external name within the load module. This operand changes the base for both symbolic and relative addresses. The two names are separated by a period. The load module name may be a member name of a partitioned data set or an alias. The entry name is the name that is duplicated in another module of the load module.

.entry
specifies an external name within a previously specified load module that you are now testing.

TCB(address)
specifies the address of a task control block (TCB). This operand is necessary when programs of the same name are assigned to two or more subtasks and you must establish uniquely which one is to be qualified, or when the load module request block is not in the TCB chain.

Example 1

Operation: Establish a base location for relative addresses to a symbol within the currently qualified program.

Known: The base address..... QSTART

```
QUALIFY QSTART
```

Example 2

Operation: Change the base location for symbolic and relative addresses to a different CSECT in the program.

Known: The module name..... PROFITS
The entry name (CSECT)..... SALES
The TCB address..... +124%

```
QUALIFY PROFITS.SALES TCB(+124%)
```

Example 3

Operation: Change the base location for relative addresses to an absolute address.

Known: The absolute address of the new base..... 5F820

```
QUALIFY 5F820.
```

Use the RUN subcommand to cause the program that is being tested to execute to termination without recognizing any breakpoints. When you specify this subcommand, TEST is terminated. When the program completes, you can enter another command. Overlay programs are not supported by the RUN subcommand. Use the GO subcommand to execute overlay programs.

SUBCOMMAND	OPERANDS
{RUN / R}	[address]

address
specifies a symbolic address, a relative address, an absolute address, or a general register containing an address. Execution will begin at the specified address. If you do not specify an address, execution begins at the last point of interruption or from the entry point if the RUN subcommand was not previously specified. (See Appendix B for more information about addresses.)

Example 1

Operation: Execute the program to termination from the last point of interruption.

```
RUN
```

Example 2

Operation: Execute a program to termination from a specific address.

Known: The address..... +A8

```
RUN  +A8
```

Use the WHERE subcommand to obtain the absolute address serving as the starting or base location for the symbolic and relative addresses in the program. Alternately, you can obtain the absolute address of an entrypoint in a particular module or control section (CSECT). If you do not specify any operands for the WHERE subcommand, you will receive the address of the next executable instruction.

SUBCOMMAND	OPERANDS
{WHERE {W	{address {load-module-name[.entryname]

address
specifies a symbolic address, a relative address, an absolute address, or a general register containing an address. When you specify an address as the operand for the WHERE subcommand, you will receive the name of the load module containing the address. (See Appendix B for more information about addresses.)

load-module-name.entry-name
specifies the name by which a load module is known, and an externally referable name within the load module. The two names are separated by a period. The load module name may be the name or an alias of a member of a partitioned data set. The entry name is the symbolic address of an entry point into the specified module. The entry name may be omitted, in which case the first entry point into the specified module will be supplied. When you specify this operand for WHERE, you will receive the main storage address of the load module.

Example 1

Operation: Obtain the absolute address of the module named CSTART.

```
WHERE CSTART
```

Example 2

Operation: Obtain the absolute address of the CSECT named JULY in the module named NETSALES.

```
WHERE NETSALES.JULY
```

Example 3

Operation: Determine in which program an absolute address is located.

Known: The absolute address..... 3E2B8

```
WHERE 3E2B8.
```

Note: You will also get the TCB address and the relative address.

Example 4

Operation: Determine the absolute address of the next executable instruction.

```
WHERE
```

TIME Command

Use the TIME command to find out how much execution time or how much session time you have used during the current session.

Program execution time is displayed when you enter the TIME command. (To enter the command while a program is executing, you must first cause an attention interruption.) Program execution time is measured from the time that the program last received input from your terminal. The TIME command has no effect upon the executing program.

Your current session time is displayed in all other instances.

COMMAND	OPERANDS
TIME	

Command Procedure Statements

A command procedure is a prearranged sequence of TSO commands and, optionally, subcommands and data. A command procedure is a convenient method for executing a repeatedly-used sequence of commands. The procedure is stored in either a data set that has CLIST as the descriptive qualifier (see the EDIT command) or in a member of a command procedure library (a pre-defined partitioned data set).

Ensure that your command specifications are complete as you will not be prompted for information while your commands are executing in a command procedure.

When using continuation characters in a command procedure, ensure that they are placed in the last usable record position. When using fixed-record format, a series of delimiters can be used to pad a record to the final position which contains the continuation character. See the paragraph entitled "Continuation of a Line in Input Mode" that appears under the EDIT command for more information on how to use the continuation character on statements in command procedures.

The statements contained in this section are designed especially for use in command procedures. They are:

- The END statement.
- The PROC statement.
- The WHEN statement.

END Statement of Command Procedures

Use the END statement to end a command procedure. When the system encounters an END statement in a command procedure, execution of the command procedure is halted and the system becomes ready to accept another command from the terminal.

STATEMENT	OPERANDS
END	

The PROC statement defines those operands in a command procedure that are to be considered as symbolic values. The user of the command procedure then supplies his own values as operands on the EXEC command. (See the discussion on the "EXEC command" for more detail on how to supply actual values for symbolic values in a command procedure.) For more information on how to assign symbolic values, refer to the publication; IBM System/360 Operating System: Time Sharing Option, Terminal User's Guide, GC28-6763.

STATEMENT	OPERANDS
PROC	number [positional-operands] [keywords]

number
specifies the number of positional operands that follow. The number must be a decimal digit. If none of the operands are positional, you must specify a zero.

positional operands
specifies one or more positional operands.

keywords
specifies one or more keyword operands.

Example 1

Operation: Use a PROC statement to identify five symbolic operands in a command procedure.

Known: Three positional operands to define..... &NAME
&NUMBER
&TIME
Two keyword operands to define..... &XREF
&MAP

```
PROC 3 NAME NUMBER TIME XREF MAP
```

Example 2

Operation: Use all three types of operands for a PROC statement.

Known: You are creating a command procedure that will use two existing programs named USERJWS.LOAD(SALESRPT) and INVENTRY.A to produce a sales report and to update the inventory. The name of the command procedure is REPORTS. You want to use different data sets as input to the procedure. The output of the first program SALESRPT will be the input for INVENTRY. You want to be able to have the output displayed at your terminal or directed to a data set so that it can be retrieved at some later date. The commands in the procedure are:

```
ALLOCATE DATASET(&LASTOUT.)  NEW BLOCK(80) SPACE(500 10)
ALLOCATE DATASET(&INPUT.)  OLD
ALLOCATE DATASET(&OUTIN.)  &NEW BLOCK(80) SPACE(500 10)
CALL (SALESRPT) '&INPUT &OUTIN.'
WHEN SYSRC(GT 4) END
CALL 'INVENTRY.A' '&OUTIN &LASTOUT.'
END
```

The PROC statement that will precede the first ALLOCATE command is:

```
PROC 2 INPUT OUTIN LASTOUT(*) NEW
```

The EXEC command to execute this procedure and have the output displayed at your terminal will be:

```
EXEC REPORTS 'FEBSALES FEBRUARY NEW'
```

when the input data set is named FEBSALES and you want to name the output from the SALESRPT program FEBRUARY. If you want to direct the output from the procedure to a data set named FEBRPT instead of to your terminal, you would enter:

```
EXEC REPORTS 'FEBSALES FEBRUARY NEW LASTOUT(FEBRPT)'
```

In this case, the symbolic values in the command procedure will be changed to:

```
ALLOCATE DATASET(FEBRPT) NEW BLOCK(80) SPACE(500 10)
ALLOCATE DATASET(FEBSALES) OLD
ALLOCATE DATASET(FEBRUARY) NEW BLOCK(80) SPACE(500 10)
CALL(SALESRPT) 'FEBSALES FEBRUARY'
WHEN SYSRC(GT 4) END
CALL 'INVENTRY.A' 'FEBRUARY FEBRPT'
END
```

Use the WHEN statement to test return codes from programs invoked via an immediately preceding CALL or LOADGO command, and to take a prescribed action if the return code meets a certain specified condition.

STATEMENT	OPERAND
WHEN	[SYSRC(operator integer)] [END command-name]

SYSRC
specifies that the return code from the previous function (the previous command in the command procedure) is to be tested according to the values specified for operator and integer.

operator
specifies one of the following operators:

EQ or = means equal to
NE or ¬= means not equal to
GT or > means greater than
LT or < means less than
GE or >= means greater than or equal to
NG or ¬> means not greater than
LE or <= means less than or equal to
NL or ¬< means not less than

integer
specifies the four digit constant that the return code is to be compared to.

END
specifies that processing is to be terminated if the comparison is true. This is the default if you do not specify a command.

command
specifies any command name and appropriate operands. The command will be processed if the comparison is true.

Appendix A: Program Product Information

Certain functions referred to in this publication (see Appendix C) are provided through IBM Program Products, which are available from IBM for a license fee. The Program Products referred to in this publication, along with additional references to related publications, are:

Interactive Terminal Facility (ITF): PL/I and BASIC - A problem solving language processor. For more information, refer to:

- TSO Interactive Terminal Facility, PL/I and BASIC Design Objectives, GC28-6822.
- TSO Interactive Terminal Facility, PL/I and BASIC Program Product Specs, C28-6831.
- TSO Interactive Terminal Facility, PL/I Introduction, C28-6838.
- TSO Interactive Terminal Facility, PL/I Terminal User's Guide, C28-6839.
- TSO Interactive Terminal Facility, BASIC Terminal User's Guide, C28-6840.

Code and Go FORTRAN - A FORTRAN compiler designed for a very fast compile-execute sequence. For more information, refer to:

- Code and Go FORTRAN Design Objectives, GC28-6823.
- Code and Go FORTRAN Processor, Installation Reference Manual, C28-6859.
- Code and Go FORTRAN Terminal User's Guide, C28-6842.
- FORTRAN Program Products for OS and OS with TSO, General Information, GC28-6824.

FORTRAN IV (G1) - A version of the FORTRAN (G) compiler modified for the terminal environment. For more information, refer to:

- FORTRAN IV (G1) Processor Design Objectives, GC28-6845.
- FORTRAN IV (G1) Processor and TSO Prompter IRM, C28-6856.
- FORTRAN IV (G1) Compiler Logic, Y28-6856.
- FORTRAN Program Products for OS and OS with TSO, General Information, GC28-6824.

TSO FORTRAN Prompter - An initialization routine to prompt the user for options and to invoke the FORTRAN IV (G1) Processor. For more information, refer to:

- TSO FORTRAN Prompter Design Objectives, GC28-6843.
- TSO FORTRAN Prompter Program Product Specs, C28-6857.
- TSO FORTRAN Prompter Logic, Y28-6410.
- FORTRAN Program Products for OS and OS with TSO, General Information, GC28-6824.

FORTRAN IV Library (Mod 1) - Execution-time routines for List directed I/O and for PAUSE and STOP capability, available for either Code and Go FORTRAN IV (G1). For more information, refer to:

- FORTRAN IV Library (Mod 1) Design Objectives, GC28-6844.
- FORTRAN IV Library (Mod 1) Program Product Specs, C28-6850.
- FORTRAN IV Library (Mod 1) Logic, Y28-6408.
- FORTRAN IV Library (Mod 1) IRM, C28-6858.
- FORTRAN Program Products for OS and OS with TSO, General Information, GC28-6824.

Full American National Standard COBOL Version 3 - A version of the American National Standard (formerly USAS) COBOL compiler modified for the terminal environment. For more information, refer to:

- Full ANS COBOL Version 3 Design Objectives, GC28-6406.
- Full ANS COBOL Compiler Library Version 3, Program Product Design Objectives, C28-6436.
- Full ANS COBOL Version 3 IRM, C28-6432.
- Full ANS COBOL Version 3 General Information, C28-6407.
- Full ANS COBOL Version 3 OS Logic, Y28-6407.
- Full ANS COBOL Version 3 OS Programming, C28-6437.

TSO COBOL Prompter - An initialization routine to prompt the user for options and to invoke the Full ANS COBOL Version 3 Processor. For more information, refer to:

- TSO COBOL Prompter Design Objectives, GC28-6404.
- TSO COBOL Prompter Logic, Y28-6406.
- TSO COBOL Prompter IRM, C28-6434.
- TSO COBOL Prompter Terminal User's Guide and Reference, C28-6433.

TSO Assembler Prompter - An initialization routine to prompt the user for options and to invoke the Assembler (F). For more information, refer to:

- TSO Assembler Prompter Design Objectives, GC26-3734.
- TSO Assembler Prompter Program Logic, Y26-3737.
- TSO Assembler Prompter User's Guide, C26-3740.

TSO Data Utilities: COPY, FORMAT, LIST, MERGE - A set of commands (and EDIT subcommands) to manipulate data sets and to format text. For more information, refer to:

- TSO Data Utilities - COPY, FORMAT, LIST, MERGE Design Objectives, GC28-6750.
- TSO Data Utilities: COPY, FORMAT, LIST, MERGE System Information, C28-6767.
- TSO Data Utilities: COPY, FORMAT, LIST, MERGE User's Guide and Reference, C28-6765.

Appendix B: Addressing Conventions Used With TEST

An address used as an operand for a subcommand of TEST may be a symbolic address, a relative address, an absolute address, or a register which may contain an address.

A symbolic address consists of one through eight alphameric characters, the first of which is an alphabetic character. The symbolic address must correspond to a symbol in the program that is being tested. Symbols cannot be used if the program being tested is a member of a partitioned data set that is part of a LINK library list unless the partitioned data set is named SYS1.LINKLIB or is the first one in the list, or unless the program is brought into main storage by TEST as an operand of the TEST command or a subsequent load command. A relative address is a hexadecimal number preceded by a plus sign (+). An absolute address is a hexadecimal number followed by a period.

Address Modifiers: An expression consisting of one of the above address types followed by a plus or a minus displacement value is also a valid address. The plus or minus displacement value can be expressed in either decimal or hexadecimal notation, as follows:

address +14n	specifies the location that is 14 bytes past that designated by "address."
address +14	specifies the location that is 20 bytes past that designated by "address."

Note: Decimal displacement (either plus or minus) is indicated by the n following the numerical offset.

Qualified Addresses: You can qualify symbolic and relative addresses to indicate that they apply to a particular control section (CSECT). To do this, you precede the address by either the name of the load module and the name of csect or just the name of csect. The qualified address must be in the form:

```
.csectname.address

or

loadname.csectname.address
```

For instance, if the user supplied name of the load module is OUTPUT, the name of the csect is CTSTART, and the symbolic address is TAXRTN you would specify:

```
.CTSTART.TAXRTN

or

OUTPUT.CTSTART.TAXRTN
```

If you do not include qualifiers, the system assumes that the address applies to the current control section.

General Registers: You can refer to a general register using the LIST or Assignment of Values subcommands by specifying a decimal integer followed by an R. The decimal integer indicates the number of the register and must be in the range zero through 15. The contents of the registers are hexadecimal characters. Other references to the general

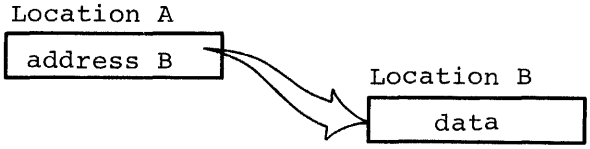
registers imply indirect addressing. The term indirect general register is used to refer to the general registers when they are used for indirect addressing.

Floating-Point Registers: You can refer to a floating-point register using the LIST or Assignment of Values subcommand by specifying a decimal integer followed by an E or a D. An E indicates a floating-point register with single precision. A D indicates a floating-point register with double precision. The decimal integer indicates the number of the register and must be a zero, two, four, or six. You must not use floating-point registers for indirect addressing; expressions composed of references to floating-point registers followed by a plus or minus displacement value or a percent sign are invalid.

Indirect Addresses: An indirect address is an address of a location or general register that contains another address. An indirect address must be followed by a percent sign (the percent sign indicates that the address is indirect). For instance, if you want to refer to some data and the address of the data is located at address A, you can specify:

```
A%
```

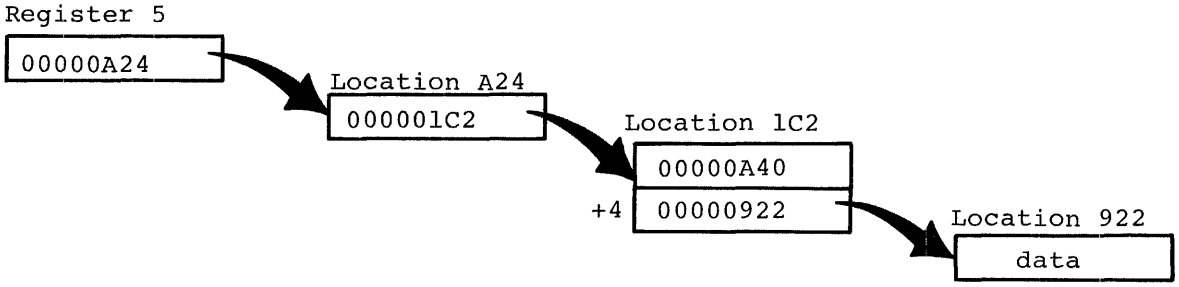
Graphically, this expression indicates:



You can indicate several levels of indirect addresses (256 levels are permitted) by following the initial indirect address with a corresponding number of percent signs. You can also include plus or minus displacement values. For instance, you may specify:

```
5R%%+4%
```

Graphically, this expression indicates:



Restriction on Symbol Use: You can refer to external symbols in a Load Module if:

- A composite external symbol dictionary (CESD), record exists.
- The TEST operand of the Link command was specified.
- The program was brought into main storage by the TEST command or one of its subtasks.

You can refer to external symbols in an Object Module if there is room in main storage for a CESD to be built.

You can refer to most internal symbols if you specify the TEST operand when you assemble and link edit your program. Exceptions are:

- Names on equate statements.
- Names on ORG, LTORG, and CNOP statements.
- Symbols more than eight bytes long.

Appendix C: Program Product Commands

ASM Command

The ASM command is provided as part of the optional TSO ASM Prompter program product which is available for a license fee. See Appendix A for IBM Program Product information.

Use the ASM command to process data sets and produce object modules. The prompter requests required information and enables you to correct your errors at the terminal.

CALC Command

The CALC command is provided as part of the optional ITF:PL/I program product which is available for a license fee. See Appendix A for IBM Program Product information.

Use the CALC command to execute ITF:PL/I statements in desk calculator mode; that is, to have statements interpreted and executed as you enter them.

COBOL Command

The COBOL command is provided as part of the optional COBOL Prompter program product which is available for a license fee. See Appendix A for IBM Program Product information.

Use the COBOL command to compile American National Standard (ANS) COBOL programs. This command reads and interprets statements for the American National Standard COBOL version 3 compiler and prompts you for any information that you have omitted or entered incorrectly. It also allocates required data sets and passes parameters to the compiler.

CONVERT Command

The CONVERT command is provided as part of the optional ITF:PL/I and BASIC program product or the Code and Go program product which is available for a license fee. See Appendix A for IBM Program Product information.

Use the CONVERT command to convert language statements contained in data sets to a form suitable for a compiler other than the one for which they were originally intended. The conversions that can be accomplished with this command are:

FROM	TO
Statements suitable for the TSO ITF:PLI compiler (a Program Product)	Statements suitable for the PL/I (F) compiler
Free format statements suitable for the Code and Go FORTRAN compiler (a Program Product)	Fixed format statements suitable for the FORTRAN (G1) compiler and all the FORTRAN compilers provided with the Operating System
Fixed format statements suitable for the FORTRAN (GI) compiler	Free format statements suitable for the Code and Go FORTRAN compiler (a Program Product)
Statements in an ITF/OS collection	A form acceptable by TSO

COPY Command

The COPY command is provided as part of the optional TSO Data Utilities: COPY, FORMAT, LIST, MERGE program product which is available for a license fee. See Appendix A for IBM Program Product information.

Use the COPY command to copy sequential or partitioned data sets. You can also use this command to:

- Add members to or merge partitioned data sets.
- Resequence line numbers of copied records.
- Change the record length, the block size, and the record format when copying into a sequential data set.

FORMAT Subcommand of EDIT

The FORMAT subcommand is provided as part of the optional TSO Data Utilities: COPY, FORMAT, LIST, MERGE program product which is available for a license fee. Appendix A contains additional information about program products.

Use the FORMAT subcommand to format textual output. This subcommand provides the facilities to:

- Print a heading on each page.
- Center lines of text between margins.
- Control the amount of space for all four margins.
- Justify left and right margins of text.
- Number pages of output consecutively.
- Halt printing when desired.

- Print multiple copies of selected pages.
- Control line and page length.
- Control paragraph indentation.

MERGE Subcommand of EDIT

The MERGE subcommand is provided as part of the optional TSO Data Utilities: COPY, FORMAT, LIST, MERGE program product which is available for a license fee. Appendix A contains additional information about program products.

Use the MERGE subcommand to:

- Merge, into the data set being edited, all or part of itself.
- Merge, into the data set being edited, all or part of another data set.

FORMAT Command

The FORMAT command is provided as part of the optional TSO Data Utilities: COPY, FORMAT, LIST, MERGE program product which is available for a license fee. See Appendix A for IBM Program Product information.

Use the FORMAT command to format textual output. This command provides the facilities to:

- Print a heading on each page.
- Center lines of text between margins.
- Control the amount of space for all four margins.
- Justify left and right margins of text.
- Number pages of output consecutively.
- Halt printing when desired.
- Print multiple copies of selected pages.
- Control line and page length.
- Control paragraph indentation.
- Store a data set that has already been formatted.
- Print all or part of a sequential or partitioned data set.

FORT Command

The FORT command is provided as part of the optional FORTRAN Prompter program product which is available for a license fee. See Appendix A for IBM Program Product information.

Use the FORT command to compile a FORTRAN IV (G1) program. You will be prompted for any information that you have omitted or entered incorrectly. It also allocates required data sets and passes parameters to the FORTRAN IV (G1) compiler.

LIST Command

The LIST command is provided as part of the optional TSO Data Utilities: COPY, FORMAT, LIST, MERGE program product which is available for a license fee. See Appendix A for IBM Program Product information.

Use the LIST command to display a sequential data set or a member of a partitioned data set. You can arrange fields within records for output; you can include or suppress record numbers; you can list all or part of a particular line of data, and you can list a single line of data, a group of lines, or a whole data set.

MERGE Command

The MERGE command is provided as part of the optional TSO Data Utilities: COPY, FORMAT, LIST, MERGE program product which is available for a license fee. See Appendix A for IBM Program Product information.

Use the MERGE command to:

- MERGE a complete or part of a sequential or member of a partitioned data set into a sequential or member of a partitioned data set.
- Copy a complete or part of a sequential or member of a partitioned data set into a new or (pre-allocated) empty sequential data set.
- Copy a complete or part of a sequential or member of a partitioned data set into a new member of an existing partitioned data set.
- Copy a complete or part of a sequential or member of a partitioned data set into a new or (pre-allocated) empty partitioned data set.

Glossary

This abbreviated glossary is a supplement to the publication: IBM Data Processing Glossary, GC20-1699. The following entries are definitions of terms used herein that are not included in the IBM Data Processing Glossary.

File name: A name of a collection of data (the file name corresponds to the data definition name).

LOGOFF: The TSO command that terminates a user's terminal session.

LOGON: The TSO command that a user must enter to initiate a terminal session.

LOGON procedure: A cataloged procedure that is executed as a result of a user entering the LOGON command.

National characters: The characters #, $, and @.

Storage dump: A recording of the contents of main or auxiliary storage so that it can be examined by a programmer or operator. (See also "dump.")

User identification: A one-to-seven character symbol identifying each TSO user.

User profile: A set of characteristics that define a TSO user to the system. Each user profile is kept in the user profile table (UPT) which in turn is stored in the user attribute data set (UADS).

Index

Indexes to systems reference library manuals are consolidated in the publication IBM System/360 Operating System: Systems Reference Library Master Index, GC28-6644. For additional information about any subject listed below, refer to other publications listed for the same subject in the Master Index.

Index

Index

IBM System/360 Operating System:
Time Sharing Option
Command Language Reference

GC28-6732-3

READER'S COMMENT FORM

Your views about this publication may help improve its usefulness; this form will be sent to the author's department for appropriate action. Using this form to request system assistance or additional publications will delay response, however. *For more direct handling of such requests, please contact your IBM representative or the IBM Branch Office serving your locality.*

Possible topics for comment are:

Clarity Accuracy Completeness Organization Index Figures Examples Legibility

What is your occupation? _____
Number of latest Technical Newsletter (if any) concerning this publication: _____
Please indicate in the space below if you wish a reply.

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. Elsewhere, an IBM office or representative will be happy to forward your comments.

Cut or Fold Along Line

Cut or Fold Along Line

Your comments, please . . .

This manual is part of a library that serves as a reference source for system analysts, programmers, and operators of IBM systems. Your comments on the other side of this form will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Fold Fold

First Class
Permit 81
Poughkeepsie
New York

Business Reply Mail
No postage stamp necessary if mailed in the U.S.A.

Postage will be paid by:

International Business Machines Corporation
Department D58, Building 706-2
PO Box 390
Poughkeepsie, New York 12602

Fold Fold

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

System/360 OS TSO Command Language Reference Printed in U.S.A. GC28-6732-3

System/360 OS TSO Command Language Reference Printed in U.S.A. GC28-6732-3

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)