GC28-0646-4
File No. S370-39

Systems

OS/VS2 TSO
Command Language Reference

VS2 Release 3.8

Includes Selectable Units:

Data Management	VS2.03.808
TSO/VTAM Level 1	VS2.03.813
System Security Support	5752-832
TSO/VTAM Level 2	5752-858

Fifth Edition (June, 1978)

This is a major revision of, and obsoletes, GC28-0646-3 and incorporates changes released in the following Selectable Unit Newsletters and System Library Supplements:

Data Management	VS2.03.808	GN28-2748	(dated July 30, 1976)
TSO/VTAM Level 1	VS2.03.813	GN28-2652	(dated May 28, 1976)
System Security Support	5752-832	GC28-0847	(dated May 27, 1977)
TSO/VTAM Level 2	5752-858	GD23-0045	(dated September 30, 1977)

See the Summary of Amendments following the Contents for a summary of the changes that have been made to this manual. A vertical line to the left of the text or illustration indicates a technical change made in this edition; revision bars are not used, however, to indicate changes made in previous editions, technical newsletters, or supplements.

This edition with Technical Newsletters GN28-4754 and GN28-4699 applies to Release 3.8 of OS/VS2 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 *System/370 Bibliography*, GC20-0001, for the editions that are applicable and current.

It is possible that this material may contain reference to, or information about, IBM products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that IBM intends to announce such IBM products, programming, or services in your country.

Publications are not stocked at the address given below; requests for 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. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Technical Newsletter

This Newsletter No. GN28-4754
Date July 30, 1980

Base Publication No. GC28-0646-4
File No. S370-39

Prerequisite Newsletters/ Supplements GN28-4699

OS/VS2 TSO Command Language Reference

This newsletter contains replacement pages for *OS/VS2 TSO Command Language Reference*.

Before inserting any of the attached pages into *OS/VS2 TSO Command Language Reference*, read **carefully** the instructions on this cover. They indicate when and how you should insert the pages.

Pages to be Removed	Attached Pages to be Inserted*
Cover - Edition Notice	Cover - Edition Notice
xi - xii	xi - xii
21 - 22	21 - 22.2
29 - 30	29 - 30
45 - 46	45 - 46
61 - 62	61 - 62
115 - 116	115 - 116
145 - 146	145 - 146
167 - 168	167 - 168
205 - 208	205 - 208
211 - 212	211 - 212
217 - 218	217 - 218
221 - 222	221 - 222.2
235 - 236	235 - 236
279 - 280	279 - 280
283 - 284	283 - 284
299 - 300	299 - 300
335 - 338	335 - 338.2

*If you are inserting pages from different Newsletters/Supplements and **identical** page numbers are involved, always use the page with the latest date (shown in the slug at the top of the page). The page with the latest date contains the most complete information.

A change to the text or to an illustration is indicated by a vertical line to the left of the change.

IBM Corporation, Publications Development, Department D58, Building 706-2
PO Box 390, Poughkeepsie, New York 12602

Printed in U.S.A.

Summary of Amendments

You can find a Summary of Amendments for this Technical Newsletter on page xi, following the Table of Contents.

Note: *Please file this cover letter at the back of the base publication to provide a record of changes.*

Preface

This publication describes the syntax and function of the commands and subcommands 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, *OS/VS2 TSO Terminal User's Guide,* GC28-0645, describes the commands 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.

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

The appendixes in the *TSO Terminal User's Guide* describe how to use the terminals supported by TSO.

The major divisions in this book are:

- Introduction
- Basic Information for Using TSO
- The Commands
- Command Procedures
- Index

The Introduction describes the TSO command language. The section entitled "Basic Information for Using TSO" contains general information necessary to use TSO commands.

The section entitled "The Commands" describes the syntax and function of each command, its operands and its subcommands. Examples are included.

The commands are presented in alphabetical order, except that the foreground-initiated background (FIB) commands are in Appendix A, the program product commands are in Appendix B, and the Access Method Services commands are in Appendix C. Subcommands are presented in alphabetical order following the command to which they apply. The END and WHEN commands, which are used with command procedures, are included in sequence instead of appearing in the Command Procedures section. Statements, variables, functions, and operators are in the Command Procedures section.

"Command Procedures" describes the control statements used in command procedures.

The publications referred to in this book are:

OS/VS2 Access Method Services, GC26-3841

OS/VS Message Library: VS2 System Messages, GC38-1002

OS/VS2 JCL, GC28-0692

OS/VS2 System Programming Library: System Generation Reference, GC26-3792

OS/VS2 System Programming Library: TSO, GC28-0629

IBM System/370 Reference Summary, GX20-1850

OS/VS Assembler Language, GC33-4010

OS/VS2 TSO Terminal Monitor Program and Service Routines Logic, SY28-0650

OS/VS2 MVS Data Management Services Guide, GC26-3875

OS/VS2 System Programming Library: Job Management, GC28-0627

Contents

Figures

Summary of Amendments for GC28-0646-4 As Updated By: GN28-4754 OS/VS2 Release 3.8

This manual has been updated to reflect service updates to OS/VS2 Release 3.8. In addition, several technical changes have been made.

Summary of Amendments for GC28-0646-4 As Updated By: GN28-4699 OS/VS2 Release 3.8

The section on the TEST command has been rewritten for technical accuracy and clarity reasons. In addition several other editorial and technical changes have been made throughout the book.

Summary of Amendments for GC28-0646-4 OS/VS2 Release 3.7

This publication contains information that was released in the following Selectable Unit Newsletters and System Library Supplements:

- OS/VS2 MVS Data Management (VS2.03.808), GN28-2748
- OS/VS2 MVS TSO/VTAM Level 1 (VS2.03.813), GN28-2652
- OS/VS2 MVS System Security Support (5752-832), GC28-0847
- OS/VS2 MVS TSO/VTAM Level 2 (5752-858), GD23-0045

The section on Command Procedures has been rewritten to present the material in a format appropriate for a reference manual. Changes in this section are not barred; therefore, you should read this section in its entirety.

Miscellaneous editorial and technical changes have been made throughout the publication. References to 2741 Communication Terminals and their use have been changed to 3270 Display Stations.

Significant technical changes have been made to the following commands and subcommands:

ALLOCATE	LISTDS
ATTRIB	LOGOFF
CALL	LOGON
EDIT	PROFILE
END (EDIT)	RENAME
FIND (EDIT)	RUN
TABSET (EDIT)	SUBMIT TERMINAL
EXEC	TEST
FREE	LIST (TEST)
LISTCAT	WHEN

Significant technical changes have been made to the following command procedure statements:

- ERROR
- OPENFILE
- TERMIN

Summary of Amendments for GC28-0646-3 As Updated by GN28-2873

Changes have been made in the following sections of this publication:

- ALLOCATE command
- CALL command
- EDIT command
- EXEC command
- FREE command
- LISTALC command
- LISTCAT command
- LISTDS command
- RUN command
- SUBMIT command
- TERMINAL command
- TEST command
- WHEN command
- OUTPUT command
- CHANGE subcommand of EDIT
- FIND subcommand of EDIT
- VERIFY subcommand of EDIT
- LIST subcommand of TEST
- QUALIFY subcommand of TEST
- Symbolic substitution examples
- Command Procedures and Program Product Commands

Summary of Amendments for GC28-0646-3 OS/VS2 Release 3.7

Changes have been made throughout this publication to reflect a Service Update to OS/VS2 Release 3.7. In addition, pertinent technical and editorial changes have been made. All references to ITF:BASIC and ITF:PLI Program Products have been deleted from this manual. As announced in P73-70, these program products have been withdrawn and reclassified to programming service classification "C" effective June 28, 1974.

Corrections have been made to the following commands:

ALLOCATE
ATTRIB
CALL
EDIT
LINK
LOGOFF
LOGON
PROFILE
RENAME
RUN
OUTPUT
SUBMIT

Corrections have been made to the following subcommands:

CHANGE (EDIT)
COPY (EDIT)
END (EDIT)
RENUM (EDIT)
RUN (EDIT)
SAVE (EDIT)
SCAN (EDIT)
SUBMIT (EDIT)
TABSET (EDIT)
UNNUM (EDIT)
AT (TEST)
LIST (TEST)
WHERE (TEST)

Introduction

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.

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.

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 established by a command.

This reference manual describes what each command can do and how to enter a command at your terminal.

Additional commands and subcommands are available for a license fee as optional program products. Appendix B lists the program product commands and subcommands.

Appendix C lists the Access Method Services commands that are available.

Before using TSO you should know how to use:

- Terminals
- TSO commands
- System-provided aids
- Data set naming conventions

Using a Terminal

A terminal session is designed to be an uncomplicated process for a terminal user: he identifies himself to the system and then issues commands to request work from the system. As the session progresses, the user has a variety of aids available at the terminal which he can use if he encounters any difficulties.

Entering Information at the Terminal

All TSO terminals have a typewriter-like keyboard through which you enter information into the system. The features of each keyboard vary from terminal to terminal; for example, one terminal may not have a backspace key, while another may not allow for lowercase letters. The features of each terminal as they apply to TSO are described in *TSO Terminal User's Guide*. The examples in this book are addressed to a user of an IBM 3270 Display Station.

Correcting Typing Errors

If you wish to correct typing errors, you must correct them before you press the ENTER key. Move the cursor under the error and type the correct character. To replace a character with a space, move the cursor under the character and press the space bar.

Using TSO Commands

A command consists of a command name followed, usually, by one or more operands. 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:

RENAME	OLDNAME	NEWNAME
command name	operand-1 (old data-set-name)	operand-2 (new data-set-name)

Two types of operands are used with the commands: *positional* and *keyword*.

Positional Operands

Positional operands follow the command name in a prescribed sequence. In the command descriptions within this manual, the positional operands are shown in lowercase characters. A typical positional operand is:

```
data-set-name
```

You must replace "data-set-name" with an actual 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 uppercase characters. A typical keyword is:

```
TEXT
```

You can specify values with some keywords. 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 "integer" when you enter the operand:

```
LINESIZE( 80 )
```

Note: If conflicting keywords are entered, the last keyword entered overrides the previous ones.

Abbreviating Keyword Operands

You can 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)

In addition, the DELETE and LISTCAT commands allow unique abbreviations for some of their keywords. They are shown with the syntax and operand descriptions of DELETE and LISTCAT.

Comments

Comments may be added to a command anywhere a blank might appear. Simply enter them within the comments delimiters /* and */. A comment may be continued to the next line by using a line continuation character (+ or -) at the end of the line.

```
listd (data-set-list) /* my data sets */
```

or

```
listd (data-set-list) /* this is a list of my -
                          active data sets */
```

Delimiters

When you type a command, you must separate the command name from the first operand by one or more blanks. You must separate operands by one or more blanks or a comma. Do not use a semicolon as a delimiter because the characters entered after a semicolon are ignored. Using a blank or a comma as a delimiter, 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.

Line Continuation

When it is necessary to continue to the next line, use a plus or minus sign as the last character of the line being worked on. Caution: a plus sign will cause leading delimiters to be removed from the continuation line.

```
list (data-set-list) /* this is a list of my -
                        active data sets */
```

or

```
alloc dataset(out.data) file(output) new +
space(10,2) tracks release
```

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 EDIT, OUTPUT, and TEST. When you enter the EDIT command, you can then enter the subcommands for EDIT. Likewise, when you enter the OUTPUT or TEST commands, you can enter the appropriate subcommands.

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 the following paragraphs.

2. You should type uppercase letters, 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. Lowercase letters, 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. Hyphens join lower-case words and symbols to form a *single* variable.

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

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

   Example: The representation

   A
   B
   C

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

6. Braces group related items, such as alternatives.

   Examples: 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).

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

   Example: The representation

   ALPHA=([A B C] ,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 only D, you may specify ALPHA=(,D).

8. 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.

Using 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 conversational messages guide you in your work at the terminal.
- The HELP command provides you with information about the commands.

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 ENTER key before typing anything else; however, input data that was being typed or output data that was being displayed 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 ENTER key.

Note: One output record from the interrupted program may be displayed at the terminal after you enter your next command. This is normal for some programs.

If your terminal has an interruption facility, you can request an attention interruption by pressing the appropriate key. 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.

Messages

There are four types of messages:

- Mode messages
- Prompting messages
- Informational messages
- Broadcast messages

Mode Messages

A mode message tells you when the system is ready to accept a new command or subcommand. When the system is ready to accept a new command it displays:

```
READY
```

When you enter a command that has subcommands and the system is ready to accept that command's subcommands, it displays the name of the command, which can be one of the following:

```
EDIT
OUTPUT
TEST
```

You can then enter the subcommands you want to use. The TEST message also appears after each TEST subcommand has been processed. If the system has to display any output or other messages, as a result of the previous command or TEST subcommand, it does so before displaying the mode message.

Sometimes you can save a little time by entering two or more commands in succession without waiting for the intervening READY message. The system then prints the READY messages in succession after the commands. If you enter the following commands without waiting for the intervening mode messages, your display will be:

```
READY
delete...
free...
rename...
READY
READY
READY
```

There is a drawback to entering commands without waiting for the intervening mode messages. If you make a mistake in one of the commands, the system sends you messages telling you of your mistake, and then it cancels the remaining commands you have entered. After you correct the error, you have to reenter the other commands.

Unless you are sure that there are no mistakes in your input, you should wait for a READY message before entering a new command.

Note: Some terminals "lock" the keyboard after you enter a command, and therefore you cannot enter commands without waiting for the intervening READY message. Terminals which do not lock the keyboard may occasionally do so, for example when all buffers allocated to the terminal are used. See *TSO Terminal User's Guide* for information on your terminal.

Prompting Messages

A prompting message tells you that required information is missing or that information you supplied was incorrectly specified. A prompting message asks you to supply or correct that information. For example, partitioned-data-set-name is a required operand of the CALL command; if you enter the CALL command without that operand the system will prompt you for the data-set-name and your display will look as follows:

```
READY
call
ENTER DATA SET NAME -
```

You should respond by entering the requested operand, in this case the data set name, and by pressing the ENTER key to enter it. For example, if the data set name is ALPHA.DATA, you would complete the prompting message as follows:

```
ENTER DATA SET NAME-
alpha.data
```

If you wish, you will receive prompting messages when appropriate. However, the PROFILE command can be used to suppress prompting.

Sometimes you can request another message that explains the initial message more fully. If the second message is not enough, you can request a further message to give you more detailed information. An indication that a second or additional message level is available is a plus sign (+) at the end of the message.

To request an additional level of message:

1. Type a question mark(?) in the first position of the line.
2. Press the ENTER key.

If you enter a question mark, and there are no messages to provide further detail, you receive the following message:

```
NO INFORMATION AVAILABLE
```

You can stop a prompting sequence by entering the requested information or by requesting an attention interruption to cancel the command.

Informational Messages

An informational message tells you about the status of the system and your terminal session. For example, an informational message can tell you how much time you have used. Informational messages do not require a response.

If an informational message ends with a plus sign (+), you can request an additional message by entering a question mark (?) after READY, as described in "Prompting Messages." Informational messages have only one second level message, while prompting messages may have more than one.

Broadcast Messages

Broadcast messages are messages of general interest to users of the system. Both the system operator and any user of the system can send broadcast messages. The system operator can send messages to all users of the system or to individual users. For example, he may send the following message to all users:

```
DO NOT USE TERMINALS #4, 5 AND 6 ON 6/30. THEY ARE
RESERVED FOR DEPARTMENT 791.
```

You, or any other user, can send messages to other users or to the system operator. For example, you may send, or receive, the following message:

```
DEPARTMENT NO. 4672 WILL BE CHANGED TO 4675 on 8/15
```

A message sent by another user will show his user identification so you will know who sent you the message.

Using the HELP Command

The HELP command can be used by a terminal user to receive all the information necessary to use any TSO command. The information requested will be displayed at the user's terminal.

Explanations of Commands

To receive a list of all the TSO commands in the SYS1.HELP data set along with a description of each, enter the HELP command as follows:

```
help
```

Information about installation-written commands may be placed in the SYS1.HELP data set. You can also get all the information available on a specific command in SYS1.HELP by entering the specific command name as an operand on the HELP command, as follows:

```
help command-name
```

Syntax Interpretation of HELP Information

The syntax notation used to present HELP information is different from the syntax notation used in this publication because it is restricted to characters that can be printed by your terminal. You can get the syntax interpretation by entering the HELP command as follows:

```
READY
help help
```

Explanations of Subcommands

When HELP exists as a subcommand, you may use it to obtain a list of subcommands or additional information about a particular subcommand. The syntax of HELP as a subcommand is the same as the HELP command.

Using Data Set Naming Conventions

A data set is a collection of related 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 topics:

1. Describe data set names in general.
2. Define the names that conform to the naming conventions for TSO.
3. Tell 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 $, @, 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.

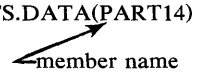
A simple data set name with only one field may be:

```
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:



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-prefix (required; defaults to userid; may be redefined using PROFILE command).
2. A user-supplied name (optional for a partitioned data set).

3. A descriptive qualifier (required).

Normally all three names are used:

USER-PREFIX.USER-SUPPLIED-NAME.DESCRIPTIVE QUALIFIER

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

WRRID.PARTS.DATA

user-prefix - WRRID
user-supplied name - PARTS
descriptive qualifier - DATA

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:

WRRID.PARTS.DATA(PART14)

User-Prefix: The user-prefix is always the leftmost qualifier of the full data set name. For TSO, this qualifier is the prefix selected in the PROFILE command. If no prefix has been selected, the userid assigned to you by your installation will be used.

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 1.

Descriptive Qualifier	Data Set Contents
ASM	Assembler (F) input
CLIST	TSO commands
CNTL	JCL and SYSIN for SUBMIT command
COBOL	American National Standard COBOL statements
DATA	Uppercase text
FORT	FORTRAN (Code and Go, G1, H) 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), PL/I Checkout, or PL/I Optimizing compiler statements.
TESTLIST	Output listing from TEST command
TEXT	Uppercase and lowercase text
VSBASIC	VSBASIC statements

Figure 1. 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:

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

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

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.) Figure 2 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: Member names must be enclosed in parentheses to distinguish them from data set names.

Figure 2. Default Names Supplied by the System

Command	Descriptive Qualifiers Input	Output	Listing
ASM	ASM	OBJ	LIST
CALL	LOAD	—	—
COBOL	COBOL	OBJ	LIST
CONVERT	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	—	—
	COBOL	—	—
SUBMIT	CNTL	—	—
TEST	OBJ	—	TESTLIST
	LOAD	—	—

Figure 3. Descriptive Qualifiers Supplied by Default

Specifying Data Set Passwords

When referencing password protected data sets, you must specify the password as part of the data set name or you will be prompted for it. The password is separated from the data set name by a slash (/) and optionally, by one or more standard delimiters (tab, blank, or comma). See the discussion on "Password Data Set" that appears under the PROTECT command for non-VSAM data sets. For VSAM data sets, see DEFINE and ALTER in *OS/VS2 Access Method Services*.

page 167+

Using Commands for VSAM and Non-VSAM Data Sets

Figure 4 gives recommended commands, by function, for VSAM and non-VSAM data sets. Numbers in parentheses after the commands indicate order of preference. Program product commands are identified with an asterisk (*). Refer to *OS/VS2 Access Method Services* for commands not covered in this document.

Function	Non-VSAM	VSAM
Build lists of attributes	ATTRIB	(None)
Allocate new DASD space	ALLOCATE	DEFINE
Connect data set to terminal	ALLOCATE	ALLOCATE
List names of allocated (connected) data sets	LISTALC	LISTALC
Modify passwords	PROTECT	DEFINE,ALTER
List attributes of one or more objects	LISTDS (1) LISTCAT (2)	LISTCAT (1) LISTDS (2)
List names of cataloged data sets		
Limit by type	LISTCAT	LISTCAT
Limit by naming convention	LISTDS	LISTDS
Catalog data sets	DEFINE (1) ALLOCATE (2)	DEFINE
List contents	EDIT,LIST*	PRINT
Rename	RENAME	ALTER
Delete	DELETE	DELETE
Copy data set	COPY*	REPRO

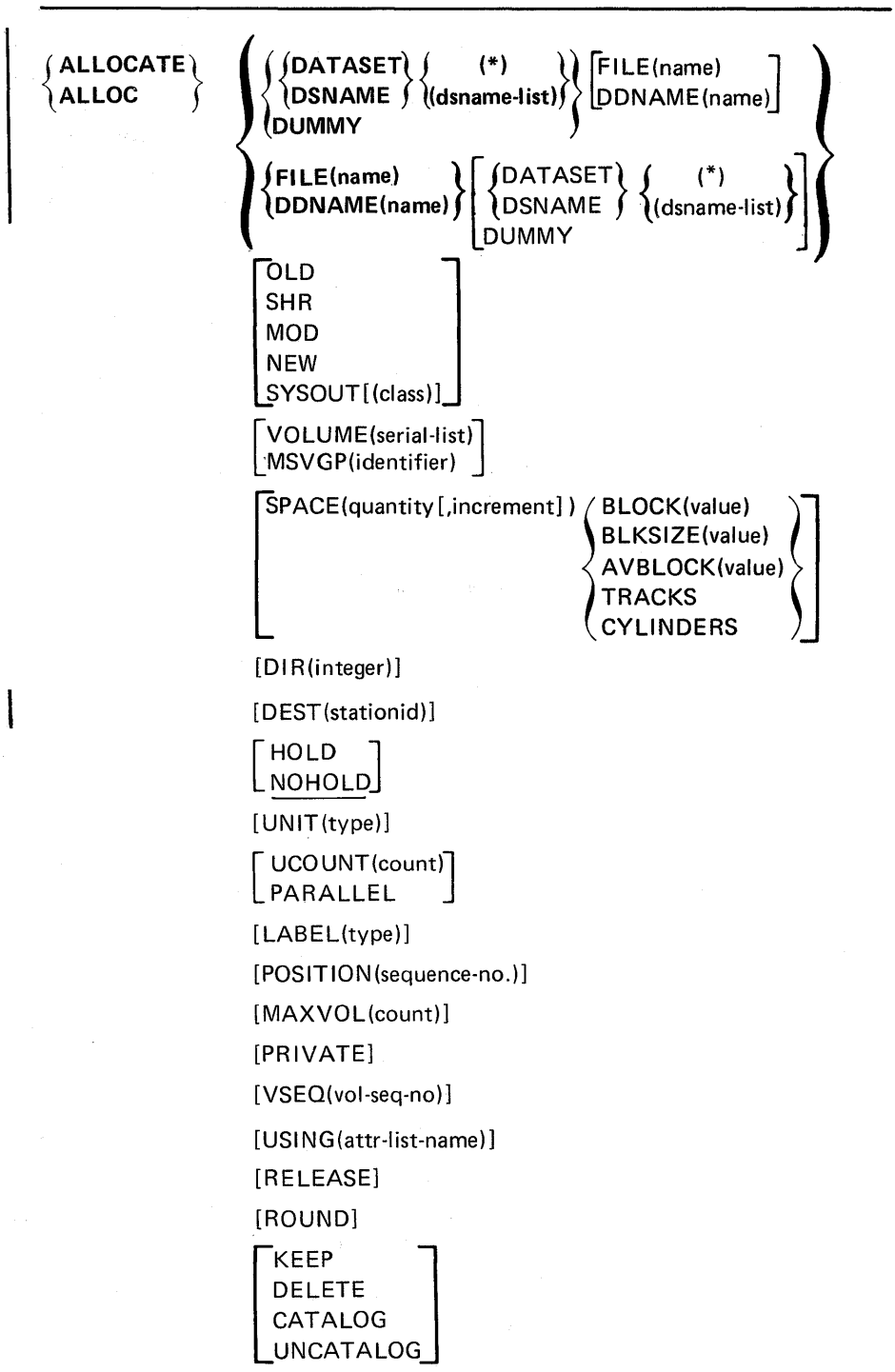
Figure 4. Commands Preferred for VSAM/Non-VSAM Data Sets

The Commands

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.

ALLOCATE Command

Use the ALLOCATE command or the ALLOCATE subcommand of EDIT (function and syntax is identical to the ALLOCATE command) to dynamically allocate the data sets required by a program that you intend to execute. You may use the ATTRIB command to build a list of attributes for non-VSAM 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 the DCB parameters for data sets being allocated.


ALLOCATE
ALLOC
DATASET
DSNAME
DUMMY
(*)
(dsname-list)
FILE(name)
DDNAME(name)
FILE(name)
DDNAME(name)
DATASET
DSNAME
DUMMY
(*)
(dsname-list)
OLD
SHR
MOD
NEW
SYSOUT[(class)]
VOLUME(serial-list)
MSVGP(identifier)
SPACE(quantity[,increment])
BLOCK(value)
BLKSIZE(value)
AVBLOCK(value)
TRACKS
CYLINDERS
[DIR(integer)]
[DEST(stationid)]
HOLD
NOHOLD
[UNIT(type)]
UCOUNT(count)
PARALLEL
[LABEL(type)]
[POSITION(sequence-no.)]
[MAXVOL(count)]
[PRIVATE]
[VSEQ(vol-seq-no)]
[USING(attr-list-name)]
[RELEASE]
[ROUND]
KEEP
DELETE
CATALOG
UNCATALOG

DATASET(dsname-list or *) or DSNAME(dsname-list or *)
specifies the name of the data set that is to be allocated. If a list of data set names is entered, ALLOCATE will allocate and concatenate non-VSAM data sets. The data set name must include the descriptive (rightmost) qualifier and may contain a member name in parentheses. If you specify a password, you will not be prompted for it when you open a non-VSAM data set. For additional information on VSAM data sets see, *OS/VS2 Access Method Services*, under the section "Data Security and Integrity."
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 or DDNAME, BLOCK or BLKSIZE, and USING operands should be entered. All other operands are ignored. No message is issued to notify the user.

Note 1: If you allocate more than one data set to your terminal, the blocksize and other data set characteristics which default on the first usage will also be used for all other data sets. This happens for input or output. The ATTRIB command and the USING keyword of ALLOCATE can be used to control the data set characteristics being used.

Note 2: 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.

Note 3: Data sets residing on the same physical tape volume cannot be allocated concurrently.

Note 4: The following items should be noted when using the concatenate function:

- The data sets specified in the list must be cataloged. You may use the CATALOG operand of either ALLOCATE or FREE to catalog a data set.
- The maximum number of data sets that can be concatenated is 255 sequential or 16 partitioned data sets. The data sets to be concatenated must be either all sequential or all partitioned.
- The data set group will be permanently concatenated. The group must be freed in order to be deconcatenated. The filename specified for the FILE or DDNAME operand on ALLOCATE must be specified for the FILE or DDNAME operand on FREE.
- All operands are ignored except for the following: DATASET/DSNAME, FILE/DDNAME, and status operands.

DUMMY
specifies that no devices or external storage space is to be allocated to the data set, and no disposition processing is to be performed on the data set. Entering the DUMMY keyword will have the same effect as specifying NULLFILE as the data set name on the DATASET or DSNAME operand. If DUMMY is specified, only the FILE or DDNAME, BLOCK or BLKSIZE, and USING operands should be entered. All other operands are ignored.

FILE(name) or DDNAME(name)
specifies the name to be associated with the data set. It may contain no more than eight characters. (This name corresponds to the name on the data definition (DD) statement in job control language and must match the ddname 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 file name 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. The DATASET or DSNAME parameter is required.

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. ALLOCATE assumes the data set is cataloged if the VOLUME operand is not entered. SHR data sets are retained by the system when you free them. The DATASET or DSNAME parameter is required.

MOD
indicates that you want to append data to the end of the data set. If the data set does not exist, a new data set is created. MOD data sets will be retained by the system when you free them. The DATASET or DSNAME parameter is required.

NEW
(non-VSAM only) indicates that the data set does not exist and that it is to be created. For new partitioned data sets you must specify the DIR operand. A NEW data set will be kept and cataloged if you specify a data set name. If you do not specify a data set name, it will be deleted when you free it or log off.

SYSOUT[(class)]
indicates that the data set is to be a system output data set. An optional subfield may be defined giving the output class of the data set. Output data will be initially directed to the job entry subsystem and may later be transcribed to a final output device. The final output device is associated with output class by the installation. After transcription by the job entry subsystem, SYSOUT data sets are deleted.

Note: If you do not specify OLD, SHR, MOD, NEW or SYSOUT, a default value is assigned, or a value is prompted for, depending on the other operands specified:

1. If any space parameters (SPACE, DIR, BLOCK, BLKSIZE, AVBLOCK, TRACKS or CYLINDERS) are specified, then the status defaults to NEW.

2. If *none* of the space parameters are entered, and the DATASET/DSNAME parameter is entered, then the status defaults to OLD.

3. If neither the DATASET or DSNAME parameter is specified or any space parameters, then you are prompted to enter a value for status.

VOLUME(serial-list)
specifies the serial number(s) of an eligible direct access volume(s) on which a new data set is to reside or on which an old data set is located. If VOLUME is specified for an old data set, the data set must be on the specified volume(s) for allocation to take place. If you do not specify VOLUME, new data sets are allocated to any eligible direct access volume. Eligibility is determined by the UNIT information in your procedure entry in the user attribute data set (UADS).

MSVGP(identifier)
specifies an installation-defined group of MSS volumes to be used for system selection of a volume or volumes to be mounted. This keyword is used for new data set allocation on MSS (3330V) devices only. It is ignored for old data sets, DUMMY, SYSOUT and terminal data sets. The user's UADS data set must contain the MOUNT attribute. Use of this keyword implies PRIVATE.

SPACE(quantity, increment)
specifies the amount of space to be allocated for a new data set. If this parameter or the primary space quantity is omitted, the default space is (10,50) AVBLOCK (1000). To indicate the unit of space for allocation, you must specify one of the following: BLOCK(value) or BLKSIZE(value), AVBLOCK(value), TRACKS, or CYLINDERS. The amount of space requested is determined as follows:

BLOCK(value) or BLKSIZE(value)
- Multiply the value of the BLOCK/BLKSIZE operand by the "quantity" value of the SPACE operand.

AVBLOCK(value)
- Multiply the value of the AVBLOCK operand by the "quantity" value of the SPACE operand.

TRACKS
- The "quantity" value of the SPACE operand is the number of tracks you are requesting.

CYLINDERS
- The "quantity" value of the SPACE operand is the number of cylinders you are requesting.

SPACE may be specified for SYSOUT, NEW, and MOD data sets. You must specify a unit of space when you use the SPACE operand.

quantity
specifies the number of units of space to be allocated initially for a data set.

increment
specifies the number of units of space to be added to the data set each time the previously allocated space has been filled.

BLOCK(value) or BLKSIZE(value)
specifies the average length (in bytes) of the records that will be written to the data set. The block value will be the unit of space used by the SPACE operand. You may specify BLOCK (value) or BLKSIZE(value) for SYSOUT, NEW, MOD, DUMMY, or terminal data sets if the default value is not acceptable. If BLKSIZE (80) is specified and RECFM=U, then the line will be truncated by 1 character. (That position (the last byte) is reserved for an attribute character.)
The following operands are ignored for SYSOUT data sets: MSVGP, VOLUME, PRIVATE, LABEL, MAXVOL, DIR, USE Q, and DISP.

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

AVBLOCK(value)
specifies only the average length (in bytes) of the records that will be written to the data set.

TRACKS
specifies that the unit of space is to be a track.

CYLINDERS
specifies that the unit of space is to be a cylinder.

Note: The keywords BLOCK, BLKSIZE, AVBLOCK, TRACKS and CYLINDERS may be specified for SYSOUT, NEW or MOD data sets. The keywords BLOCK or BLKSIZE can also be specified for dummy or terminal data sets.

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.

DEST(stationid)
specifies a remote work station to which SYSOUT data sets will be directed upon unallocation. The stationid is the one to seven character name of the remote work station receiving the SYSOUT data set.

HOLD
specifies that the data set is to be placed on a HOLD queue upon unallocation.

NOHOLD
specifies that processing of the output should be determined via the HOLD/NOHOLD specification associated with the particular SYSOUT class specified. However, the specification associated with the SYSOUT class may be overridden by using the NOHOLD keyword on the FREE command.

UNIT(type)

specifies the unit type to which a file or data set is to be allocated. You may specify an installation-defined group name, a generic device type, or a specific device address. If volume information is not supplied, (volume and unit information is retrieved from a catalog) the unit type that is coded will override the unit type from the catalog. This condition exists only if the coded type and class are the same as the cataloged type and class.

UCOUNT(count)
specifies the maximum number of devices to be allocated, where count is a value from 1-59.

PARALLEL
specifies that one device is to be mounted for each volume specified on the VOLUME operand or in the catalog.

LABEL(type)
specifies the kind of label processing to be done. Type may be one of the following:
SL, SUL, AL, AUL, NSL, NL, LTM, or BLP. These types correspond to the present JCL label-type values.

POSITION(sequence-no.)
specifies the relative position (1-9999) of the data set on a multiple data set tape. The sequence number corresponds to the data set sequence number field of the label parameter in JCL.

MAXVOL(count)
specifies the maximum number (1-255) of volumes a data set can use. This number corresponds to the count field on the VOLUME parameter in JCL.

PRIVATE
specifies that the private volume use attribute be assigned to a volume that is not reserved or permanently resident. This operand corresponds to the PRIVATE keyword of the VOLUME parameter in JCL.

Note: If VOLUME and PRIVATE operands are not specified and the value specified for MAXVOL exceeds the value specified for UCOUNT, the system will not demount any volumes when all of the mounted volumes have been used, causing abnormal termination of your job. If PRIVATE is specified, the system will demount one of the volumes and mount another volume in its place so that processing can continue.

VSEQ(vol-seq-no.)
specifies at which volume (1-255) of a multi-volume data set processing is to begin. This operand corresponds to the volume sequence number on the VOLUME parameter in JCL. VSEQ should only be specified when the data set is cataloged.

USING(attr-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 users 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 ATTRIB command allocates a file with the name being the (attr-list-name) specified in 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(attr-list-name) operand.

RELEASE
specifies that unused space is to be deleted when the data set is freed.

Note: If RELEASE is used with a new data set with the BLOCK or BLKSIZE parameter, then the SPACE parameter must be used.

ROUND
specifies that the allocated space be equal to one or more cylinders. This operand should be specified only when space is requested in units of blocks. This operand corresponds to the ROUND keyword on the SPACE parameter in JCL.

Note: The final disposition of the following operands can be modified by a command processor.

KEEP
specifies that the data set is to be retained by the system after it is freed.

DELETE
specifies that the data set is to be deleted after it is freed.

CATALOG
specifies that the data set is to be retained by the system in a catalog after it is freed.

UNCATALOG
specifies that the data set is to be removed from the catalog after it is freed. The data set is still retained by the system.

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: MOSER7.INPUT.DATA

```
allocate dataset(input.data) old
```

Example 2

Operation: Allocate a new data set.

Known:
The name that you want to give the data: MOSER7.OUTPUT.DATA
The number of tracks expected to be used: 10
DCB parameters are in an attribute list named ATTR.

```
allocate dataset(output.data) new space(10,2) tracks
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: MOSER7.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)
```

Example 6

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

Known:
The data set name: MOSER7.OUT.DATA
The file name: OUTPUT
You don't want to hold unused space.

```
alloc dataset(out.data) file(output) new space(10,2)
tracks release
```

Example 7

Operation: Allocate an existing multi-volume data set to SYSDA, with one device mounted for each volume.

Known:

Data set name -	MOSER7.MULTIVOL.DATA
volumes -	D95VL1
	D95VL2
	D95VL3
filename -	SYSLIB

```
alloc dataset('moser7.multivol.data') old parallel
file(syslib) volume(d95vl1,d95vl2,d95vl3)
unit(sysda)
```

Example 8

Operation: Allocate an existing data set on the second file of a standard-label tape.

Known:

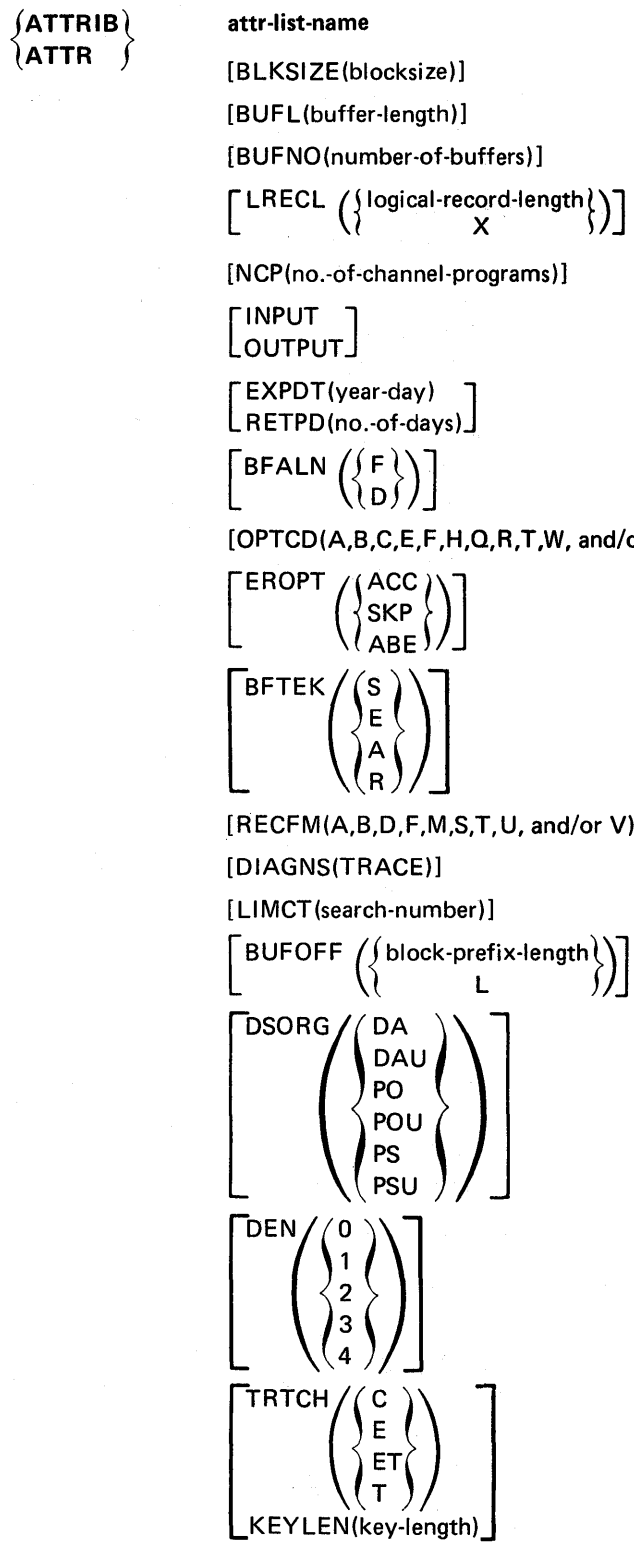
Data set name -	MOSER7.TAPE1.DATA
volume -	TAPEVL
unit -	2400

```
alloc dataset('moser7.tape1.data') label(sl)
unit(2400) volume(tapevl) position(2)
```

ATTRIB Command

Use the ATTRIB command to build a list of attributes for non-VSAM 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 and LABEL parameters for data sets being allocated. See also the subparameters of the DCB parameter in *OS/VS2 JCL*.

The ATTRIB command allocates a file with the same name as your attribute-list-name. You can use the LISTALC command with the STATUS keyword to list your active attribute lists. The data set name is NULLFILE which is also the data set name for files allocated with the DUMMY keyword of the ALLOCATE command. Note that, since this is a NULLFILE allocation, it is subject to use and modification by other commands. Therefore, it is advisable to allocate those data sets for which the attribute list was built before you issue any commands that may cause NULLFILE allocation, such as LINK or RUN.


ATTRIB
ATTR
attr-list-name
[BLKSIZE(blocksize)]
[BUFL(buffer-length)]
[BUFNO(number-of-buffers)]
LRECL
logical-record-length
X
[NCP(no.-of-channel-programs)]
INPUT
OUTPUT
EXPDT(year-day)
RETPD(no.-of-days)
BFALN
F
D
[OPTCD(A,B,C,E,F,H,Q,R,T,W, and/or Z)]
EROPT
ACC
SKP
ABE
BFTEK
S
E
A
R
[RECFM(A,B,D,F,M,S,T,U, and/or V)]
[DIAGNS(TRACE)]
[LIMCT(search-number)]
BUFOFF
block-prefix-length
L
DSORG
DA
DAU
PO
POU
PS
PSU
DEN
0
1
2
3
4
TRTCH
C
E
ET
T
KEYLEN(key-length)

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(blocksize)
specifies the block size 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 four byte block descriptor word in addition to the largest logical record length. The logical record length must allow space for a four-byte record descriptor word.)
- RECFM(V B), then the block size must be equal to or greater than the largest block in the data set. (Note: For block variable length records, the size of the largest block must allow space for the four 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 four-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.
- RECFM(U) and BLKSIZE(80), then one character will be truncated from the line, that character (the last byte) is reserved for an attribute character.

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.

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 four 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 (VS or VBS) processed by QSAM (locate mode) or BSAM, specify LRECL (X) when the logical record exceeds 32,756 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 may be a one to 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(A,B,C,E,F,H,Q,R,T,W and/or Z)
specifies the following optional services that you want the system to perform. (See also the OPTCD subparameter of the DCB parameter in *OS/VS2 JCL* for a detailed discussion of these services.)

A specifies that actual device addresses be presented in READ and WRITE macro instructions.
B specifies that end-of-file (EOF) recognition be disregarded for tapes.
C specifies the use of chained scheduling.
E requests an extended search for block or available space.
F specifies that feedback from a READ or WRITE macro instruction should return the device address in the form it is presented to the control program.
H requests the system to check for and bypass.
Q requests the system to translate a magnetic tape from ASCII to EBCDIC or from EBCDIC to ASCII.
R requests the use of relative block addressing.
T requests the use of the user totaling facility.
W requests the system to perform a validity check when data is written on a direct access device.
Z requests the control program to shorten its normal error recovery procedure for input on magnetic tape.

(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 to accept the block of records in which the error was found.
SKP to skip the block of records in which the error was found.
ABE to 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,D,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. (See also the RECFM subparameter of the DCB parameter in *OS/VS2 JCL* for a detailed discussion of the formats and characteristics.)
Use the following values with the RECFM operand.
A indicates that the record contains ASCII printer control characters.
B indicates that the records are blocked.
D indicates variable-length ASCII records.
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. Exchange 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).

DIAGNS(TRACE)
specifies the Open/Close/EOV trace option that gives a module-by-module trace of the Open/Close/EOV work area and the user's DCB.

LIMCT(search-number)
specifies the number of blocks or tracks to be searched for a block or available space. The number must not exceed 32,760.

BUFOFF({ block-prefix-length / L })
specifies the buffer offset. The block prefix length must not exceed 99. "L" is specified if the block prefix field is four bytes long and contains the block length.

DSORG({ DA / DAU / PO / POU / PS / PSU })
specifies the data set organization as follows:
DA - direct access
DAU - direct access unmovable
PO - partitioned organization
POU - partitioned organization unmovable
PS - physical sequential
PSU - physical sequential unmovable

DEN({ **0** | **1** | **2** | **3** | **4** } **)**

specifies the magnetic tape density as follows:

0 - 200 bpi/7 track
1 - 556 bpi/7 track
2 - 800 bpi/7 and 9 track
3 - 1600 bpi/9 track
4 - 6250 bpi/9 track (IBM 3420 Models 4, 6, and 8, or equivalent)

TRTCH({ **C** | **E** | **T** | **ET** } **)**

specifies the recording technique for 7-track tape as follows:

C data conversion with odd parity and no translation
E even parity with no translation and no conversion
T odd parity and no conversion; BCD to EBCDIC translation when reading and EBCDIC to BCD translation when writing
ET even parity and no conversion; BCD to EBCDIC translation when reading and EBCDIC to BCD translation when writing

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. This keyword is mutually exclusive with TRTCH.

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: chained-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 32,756 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.

CALL { dsname | dsname(membername) }

['parameter-string']

---

dsname(membername)
specifies the name of a partitioned data set and the membername (program name) to be executed. The membername must be enclosed in parentheses.

Note: A temporary tasklib is established when programs are invoked via the CALL command. The tasklib is effective for the execution of the CALL command and the tasklib data set is the same as the dsname specified on the invocation of the CALL command.

If the name of the partitioned data set does not conform to the data set naming conventions, it must include the member name in the following manner:

```
dsname(membername)
```

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

```
'wrrid.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: JUDAL.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: JUDAL.MYLIB.LOAD(COS1)

```
call mylib(cos1)
```

Example 3

Operation: Execute a load module.

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

```
call (sin1)
```

DELETE Command

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

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.

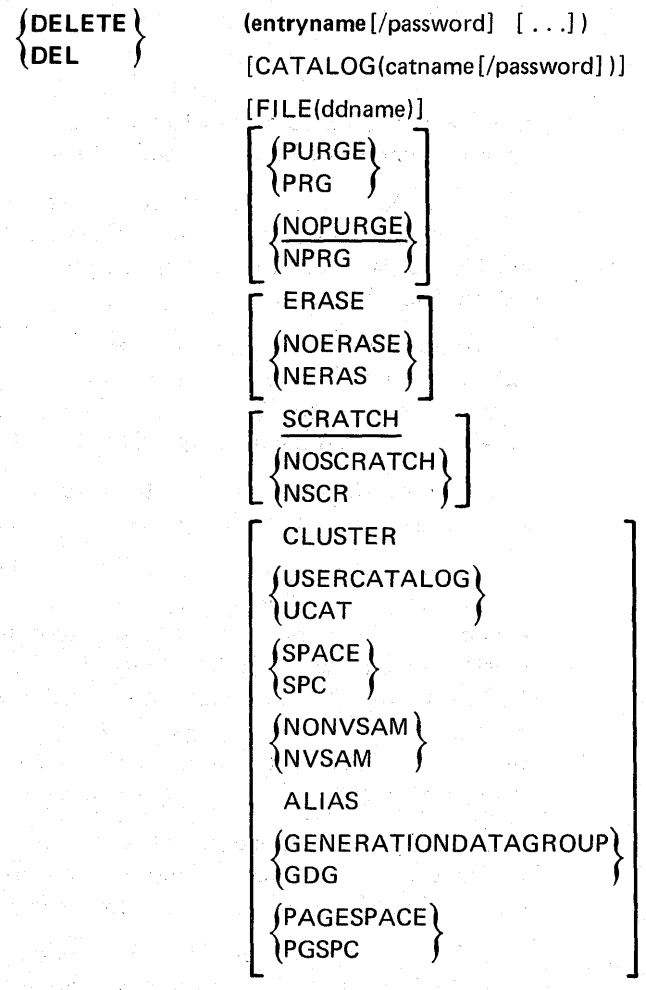
Members of a partitioned data set and aliases for any members must each be deleted explicitly. That is, when you delete a member, the system does not remove any alias names of the member; likewise, when you delete an alias name, the member itself is not deleted.

If a generation-data-group entry is to be deleted, any generation data sets that belong to it must have been deleted.

For MVS, the original TSO DELETE command has been replaced by the Access Method Services command with the same name. The explanations given below provide the information required to use these services for normal TSO operations. The TSO user who wants to manipulate VSAM objects or who wants to use the other Access Method Services from his terminal should refer to *OS/VS2 Access Method Services*. For error message information, refer to *OS/VS Message Library: VS2 System Messages*.

The DELETE command supports unique operand abbreviations in addition to the usual abbreviations produced by truncation. The syntax and operand explanations show these unique cases.

Before you delete a protected non-VSAM data set, you should use the PROTECT command to delete the password from the password data set. This will prevent your having insufficient space for future entries.



entryname[/password][...]
is a required parameter that names the entries to be deleted. When more than one entry is to be deleted, the list of entry names must be enclosed in parentheses. This parameter must be the first parameter following DELETE.

If you want to delete several data set entries having similar names, you may insert an asterisk into the data set name at the point of dissimilarity. That is, all data set entries 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 not place it in the first position. TSO does not prefix the userid when an asterisk appears in the first position.

For instance, suppose that you have several data set entries named:

```
VACOT.SOURCE.PLI
VACOT.SOURCE2.PLI
VACOT.SOURCE2.TEXT
VACOT.SOURCE2.DATA
```

If you specify:

```
delete source2.*
```

the only data set entry remaining will be

```
VACOT.SOURCE.PLI
```

password
specifies a password for a password-protected entry. Passwords may be specified for each entry name or the catalog's password may be specified through the CATALOG parameter for the catalog that contains the entries to be deleted.

CATALOG(catname[/password])
specifies the name of the catalog that contains the entries to be deleted.

catname
identifies the catalog that contains the entry to be deleted.

password
specifies the master password of the catalog that contains the entries to be deleted.

FILE(ddname)
specifies the name of the DD statement that identifies the volume that contains the data set to be deleted or identifies the entry to be deleted.

PURGE or PRG
specifies that the entry is to be deleted even if the retention period, specified in the TO or FOR parameter, has not expired.

NOPURGE or NPRG
specifies that the entry is not to be deleted if the retention period has not expired. When NOPURGE is coded and the retention period has not expired, the entry is not deleted. If neither PURGE nor NOPURGE is coded, NOPURGE is the default.

ERASE
specifies that the data component of a cluster (VSAM only) is to be overwritten with binary zeros when the cluster is deleted. If ERASE is specified, the volume that contains the data component must be mounted.

NOERASE or NERAS
specifies that the data component of a cluster (VSAM only) is not to be overwritten with binary zeros when the cluster is deleted.

SCRATCH
specifies that a non-VSAM data set is to be scratched (removed) from the volume table of contents (VTOC) of the volume on which it resides. SCRATCH is the default if neither SCRATCH nor NOSCRATCH is specified.

NOSCRATCH or NSCR
specifies that a non-VSAM data set is not to be scratched (removed) from the VTOC of the volume on which it resides.

CLUSTER
specifies that the entry to be deleted is a cluster entry for a VSAM data set.

USERCATALOG or UCAT
specifies that the entry to be deleted is a user-catalog entry. This parameter must be specified if a user catalog is to be deleted. A user catalog can be deleted only if it is empty.

SPACE
specifies that the entry to be deleted is a data-space entry. This parameter is required if a data space is to be deleted. A data space can be deleted only if it is empty.

NONVSAM or NVSAM
specifies that the entry to be deleted is a non-VSAM data set entry.

ALIAS
specifies that the entry to be deleted is an alias entry.

GENERATIONDATAGROUP or GDG
specifies that the entry to be deleted is a generation-data-group entry. A generation-data-group base can be deleted only if it is empty.

PAGESPACE or PGSPC
specifies that a page space is to be deleted. A page space can be deleted only if it is inactive.

If the FILE parameter is omitted, the entryname is dynamically allocated in the following cases:

- A non-VSAM entry is to be deleted and scratched.
- An entry is to be deleted and erased.
- An entry that resides in a data space of its own is to be deleted.

Example

Operation: Delete an entry. In this example, a non-VSAM data set is deleted:

Known:
The prefix in the user's profile is D27UCAT.
Your userid is D27UCAT.

```
delete  example.nonvsam scratch nonvsam
```

The DELETE command deletes the non-VSAM data set (D27UCAT.EXAMPLE.NONVSAM). Because the catalog in which the entry resides is assumed not to be password protected, the CATALOG parameter is not required to delete the non-VSAM entry.

SCRATCH removes the VTOC entry of the non-VSAM data set. Because FILE is not coded, the volume that contains D27UCAT.EXAMPLE.NONVSAM is dynamically allocated.

NONVSAM ensures that the entry being deleted is a non-VSAM data set. However, DELETE can still find and delete a non-VSAM data set if NONVSAM is omitted.

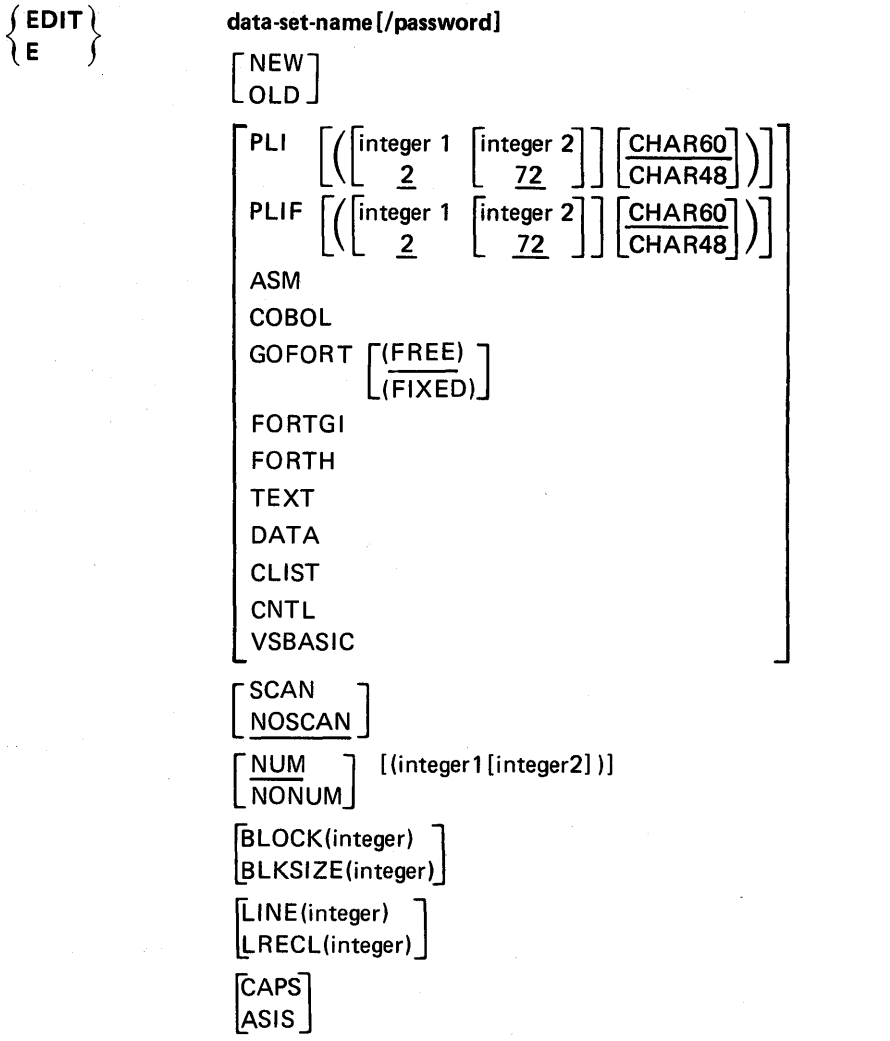
The EDIT command is the primary facility for entering data into the system. Therefore, almost every application involves some use of EDIT. With EDIT and its subcommands, you can create, modify, store, submit, retrieve, and delete data sets with sequential or partitioned data set organization. The data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

- Source programs composed of program language statements (PL/I, 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 language (JCL) statements for background jobs

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

- Fixed blocked, unblocked, or standard block; with or without ASCII and machine record formats
- Variable blocked or unblocked; without ASCII or machine control characters

EDIT support of print control data sets is "read only." Whenever a SAVE subcommand is entered for an EDIT data set orginally containing print control characters, the ability to print the data set on the printer with appropriate spaces and ejects is lost. If you enter SAVE without operands for a data set containing control characters, you will be warned that the data set will be saved without control characters, and you can elect to either save into the original data set or enter a new data set name. If the data set specified on the EDIT command is partitioned and contains print control characters, a save into it will not be allowed.



data-set-name
specifies the name of the data set that you want to create or edit.

password
specifies the password associated with the data-set-name. If the password is omitted and the data set is password protected, you will be prompted for the data set's password. Read protected partitioned data sets will cause a prompt for the password twice, provided it is not entered on the EDIT command, or is not the same password as your LOGON userid password.

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, a sequential data set is allocated 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, the system defaults to NEW.

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 5, note 4).

PLI
specifies that the data identified by the first operand is for PL/I statements that are to be held as V-format records with a maximum length of 104 bytes. The statements may be for the PL/I Optimizing compiler or the PL/I Checkout compiler.

PLIF
specifies that the data set identified by the first operand is for PL/I statements that are to be held as fixed format records 80 bytes long. The statements may be for the PL/I Optimizing compiler or the PL/I Checkout compiler.

integer1 and integer2
specify the column boundaries for your input statements. These values are applicable only when you request syntax checking of a data set for which the PLIF operand has been specified. 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 no value is entered, the default value is CHAR60.

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. The data set will be assigned line numbers.

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 or subcommand.

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.

FORTGI
specifies that the data set identified by the first operand is for FORTRAN IV (G1) statements.

FORTH
specifies that the data set identified by the first operand is for FORTRAN IV (H) EXTCOMP 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. 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 FORTGI or FORTH operand.
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.

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

Note: The ASM, CLIST, CNTL, COBOL, DATA, FORTGI, FORTH, GOFORT, PLI, PLIF, TEXT, and VSBASIC 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 (that is, it is enclosed in quotes).

- The data-set-name operand is a member name only (that is, 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 1 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, or FORTH you must enter the data set type keyword to save it.

SCAN
specifies that each line of data you enter in input mode is to be checked statement by statement for proper syntax. Syntax checking is available only for statements written in GOFORT, FORTGI, FORTH.

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 assume appropriate default values.

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

BLOCK(integer) or BLKSIZE(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 or editing an empty old data set. You cannot change the block size of an existing data set except if the data set is empty. If you omit this operand, it will default according to the type of data set being created. Default block sizes are described in Figure 5. If different defaults are established at system generation (SYSGEN) time, Figure 5 values may not be applicable. The block size (BLOCK or BLKSIZE), for data sets that contain fixed-length records must be a multiple of the record length (LINE or LRECL); for variable-length records, the block size must be a multiple of the record length plus 4.

Note: If BLKSIZE (80) is coded with RECFM(U), then the line will be truncated by 1 character. This byte (the last one) is reserved for an attribute character.

LINE(integer) or LRECL(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 or editing an empty old 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 unless the data set is empty. If you specify this operand and the data set type is ASM, 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 5. This operand is used in conjunction with the BLOCK or BLKSIZE operand.

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

ASIS
specifies that input and output data is to retain the same form (uppercase and lowercase) as entered. ASIS is the default for TEXT only.

Data Set Type	DSORG	LRECL		Block Size		Line Numbers		
		LINE(n)		BLOCK(n)		NUM (n, m)	CAPS/ASIS	
		default	specif.	default	specif. (Note 1)	default (n, m) spec.	default	CAPS Required
ASM	PS/PO	80	=80	3120	≤default	Last 8 $73 \leq n \leq 80$	CAPS	Yes
CLIST	PS/PO	255	(Note 2)	3120	≤default	(Note 3)	CAPS	Yes
CNTL	PS/PO	80	=80	3120	≤default	Last 8	CAPS	Yes
COBOL	PS/PO	80	=80	400	≤default	First 6	CAPS	Yes
DATA	PS/PO	80	≤255	3120	≤default	Last 8	CAPS	No
FORTGI	PS/PO	255	=80	400	≤default	Last 8	CAPS	Yes
FORTH	PS/PO	255	=80	400	≤default	Last 8	CAPS	Yes
GOFORT	PS/PO	255		3120	≤default	First 8	CAPS	Yes
(or user supplied data set type — See Note 4)								
PLI	PS/PO	104	≤100	400	≤default	(Note 3)	CAPS	No
PLIF	PS/PO	80	≤100	400	≤default	Last 8	CAPS	Yes
TEXT	PS/PO	255	(Note 2)	3120	≤default	(Note 3)	ASIS	No
VSBASIC	PS/PO	255	=80	3120	≤=32,760	First 5	CAPS	Yes

Notes:

1. The default or maximum allowable block size may be specified at SYSGEN time.
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.
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.
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 *OS/VS2 SPL: System Generation Reference.*

   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 Numbers: V type: First 8 characters
   F type: Last 8 characters

Figure 5. Default Values for LINE or LRECL and BLOCK or BLKSIZE Operands

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.

If you have limited access to your data set, by assigning a password, you can 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. The default is an interval of 10. Line numbers make editing much easier, because 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.

All input records will be converted to uppercase characters, except when you specify the ASIS or TEXT operand. The TEXT operand also specifies that character-deleting indicators and tabulation characters will be recognized, but all other characters will be added to the data set unchanged.

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).

You can create a variety of FORTRAN data sets: FORTGI, FORTH, and GOFORT. You can enter GOFORT input statements in "free form," that is, there are no specific columns into which your statements must go. Free form FORTRAN statements 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 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 or plus sign (that is, a hyphen or plus sign followed immediately by pressing the ENTER key). The situations are:

- The syntax checking facility is being used.
- The data set type is GOFORT(FREE).
- 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 or plus sign 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. If the command procedure 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 command procedure 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 command procedure 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 (for example, CHANGE or 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 (that is, while in subcommand mode or in input mode for other than a command procedure data set), enter a hyphen in the next-to-last column, a blank in the last column, and immediately press the ENTER key.

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 6 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
ALLOCATE	No change
BOTTOM	Last line (or zero for empty data sets)
CHANGE	Last line changed
COPY	Last line copied
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
EXEC	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
MOVE	Last line moved
PROFILE	No change
RENUM	Same relative line
RUN	No change
SAVE	No change or same relative line
SCAN	Last line scanned, if any
SEND	No change
SUBMIT	No change
TABSET	No change
TOP	Zero value
UNNUM	Same relative line
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 6. How EDIT Subcommands Affect the Line Pointer Value

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:

- You press the ENTER key before typing anything.

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.

- 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.

- You enter the INSERT subcommand with no operands.

You will switch from input mode to edit mode when:

- You press the ENTER key before typing anything.
- You cause an attention interruption.
- There is no more space for records to be inserted into the data set and resequencing is not allowed.
- 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. (See *TSO Terminal User's Guide* to determine if your terminal supports tab setting.) 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 or the SAVE operand of the END subcommand for this purpose.

Recovering Data after a Terminal Line Has Been Disconnected

If a terminal is disconnected during an EDIT session, the system will attempt to save a copy of the edited data set (with all changes) into another data set. The data set used for saving is named by applying data set naming conventions to an intermediate qualifier name of EDITSAVE. This data set can be edited when you log on again.

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: WRR05.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: WRR05.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
```

Example 3

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

Known:
The name of the data set: WRR05.MANHRS.DATA
Block size: 3120
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
```

Subcommands for EDIT

Use the subcommands while in edit mode to edit and manipulate data and to communicate with the system operator and with other terminal users. 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 7 contains a summary of each subcommand's function.

Note: For a complete description of the syntax and function of the ALLOCATE, EXEC, HELP, PROFILE, SEND, and SUBMIT subcommands, refer to the description of the TSO command with the same name.

ALLOCATE	Allocates data sets and filenames.
BOTTOM	Moves the pointer to the last record in the data set.
CHANGE	Alters the contents of a data set.
COPY	Copies records within the data set.
DELETE	Removes records.
DOWN	Moves the pointer toward the end of the data.
END	Terminates the EDIT command.
EXEC	Executes a command procedure.
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.
MOVE	Moves records within a data set.
PROFILE	Specifies characteristics of your user profile.
RENUM	Numbers or renumbers lines of data.
RUN	Causes compilation and execution of data set.
SAVE	Retains the data set.
SCAN	Controls syntax checking.
SEND	Allows you to communicate with the system operator and with other terminal users.
SUBMIT	Submits a job for execution in the background.
TABSET	Sets the tabs.
TOP	Sets the pointer to zero value.
UNNUM	Removes line numbers from records.
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 7. Subcommands of the EDIT Command

ALLOCATE Subcommand of EDIT

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

Refer to the ALLOCATE command for the description of the syntax and function of the ALLOCATE subcommand.

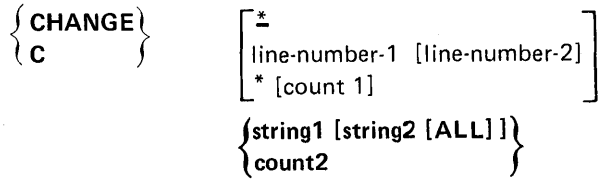
BOTTOM Subcommand of EDIT

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.

{ BOTTOM }
{ B   }

CHANGE Subcommand of EDIT

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



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 first occurrence of the sequence of characters specified for string1.

*****
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 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 (-) and plus (+) signs can be used but should be avoided due to possible confusion with their use in continuation. If the first character in the character string is an asterisk (*), do not use a slash (/) as the extra character. (TSO interprets the /* as the beginning of a comment.) 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 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 complete the line 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 delimiter must be the same as the extra character used for string1. (Optionally, this delimiter can also immediately follow string2.)

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 string 2.

If you cause an attention interruption during the CHANGE subcommand when using the ALL keyword, your data set may only be partially changed. It is good practice to list the affected area of your data set before continuing.

If the special delimiter form is used, string2 must be terminated by the delimiter 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:

- You cannot use both special-delimiter and quoted-string notation in the same subcommand.
- Each string must be enclosed with single quotes, for example, 'This is string1' 'This is string2.' Quoted strings must be separated with a blank.
- A single quote within a character string is represented by two single quotes, for example, 'pilgrim''s progress'.
- A null string is represented by two single quotes, for example, ''.

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!!or!ab!	'ab' ''
Print up to	!ab	'ab'
Place in front of	!!cde!	'' 'cde'

*—using the exclamation point (!) as the delimiter.

Note: Choose the form that best suits you (either special-delimiter or quoted-string) and use it consistently. It will help you use the subcommand.

Note: If you cause an attention interruption during the CHANGE subcommand your data set might not be completely changed. You should list the affected part of your data set before entering other subcommands.

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 chacter 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.

```
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 displayed at your terminal. You complete the line by typing the new information and enter the line by pressing the ENTER 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/
```

Example 8

Operation: Delete a sequence of characters wherever it appears in a line-numbered data set containing line numbers 10 to 150.

Known:
The sequence of characters to be deleted: weekly

```
change 10 999/ weekly// all
```

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' ''
```

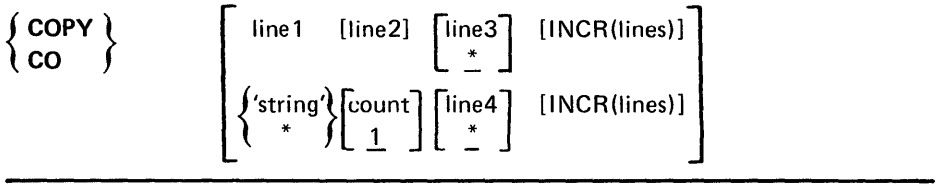
COPY Subcommand of EDIT

Use the COPY subcommand of EDIT to copy one or more records that exist in the data set being edited. The copy operation moves data from a source location to a target location within the same data set and leaves the source data intact. Existing lines in the target area are shifted toward the end of the data set as required to make room for the incoming data. No lines are lost.

The target line cannot be within the source area, with the exception that the target line (the first line of the target area) can overlap the last line of the source area.

Upon completion of the copy operation, the current line pointer points to the last copied-to line, not to the last line shifted to make room in the target area.

Note: If you cause an attention interruption during the copy operation, the data set may be only partially changed. As a check, list the affected part of the data set before continuing.



Note: If COPY is entered without operands, the line pointed to by the current line pointer is copied into the current-line + EDIT-default-increment location.

line1
specifies the first line or the lower limit of the range to be copied. If the specified line number does not exist in this data set, the range begins with the next higher line number.

line2
specifies the last line or the upper limit of the range to be copied. If the specified line number does not exist in this data set, the range ends with the highest line number that is less than line2. If line2 is not entered, the value defaults to the value of line1; that is, the source becomes one line. Do not enter an asterisk for line2.

Note: If COPY is followed by two line-number operands, the system assumes them to represent line1 and line3, and defaults line2 to the value of line1.

line3
specifies the target line number: that is, the line at which the copied-to data area will start. If the line3 value corresponds to an existing line, the target line is changed to line3 + INCR(lines) and either becomes a new line or displaces an existing line at that location. Once the copy operation begins, existing lines encountered in the target area are renumbered to make room for the incoming data. The increment for renumbered lines is one (1). Specifying zero (0) for line3 puts the copied data at the top of the data set only if line 0 is empty: if line 0 has data, enter TOP followed by COPY with line3 set to *. Note that line3 defaults to *.

Note: The value of line3 should not fall in the range from line1 to line2: that is, the target line must not be in the range being copied. Exception: line3 can be equal to line2.

*****
represents the value of the current line pointer.

INCR(lines)
specifies the line number increment to be used for this copy operation. The default is the value in effect for this data before the copy operation. When the copy operation is complete, the increment reverts to the value in effect before COPY was issued. Range: 1-8 decimal digits but not zero.

Note: The increment for any renumbered lines is one (1).

'string'
specifies a sequence of alphameric characters with a maximum length equal to or less than the logical record length of the data set being edited. When a character string is specified, a search starting at the current line is done for the line containing the string. When found, that line is the start of the range to be copied for either numbered or unnumbered data sets.

count
specifies the total number of lines (the range) to be copied. The default for count is one (1). Enter 1-8 decimal digits but not zero (0) or asterisk (*).

line4
applies to both numbered and unnumbered data sets. For unnumbered data sets, line4 specifies the target line (the line at which the copied-to data area will start) as a relative line number (the nth line in the data set). For numbered data sets, line4 is specified the same as line3. Specifying zero (0) for line4 puts the copied data at the top of the data set only if line (0) is empty; if line (0) has data, enter TOP followed by COPY with line4 set to *. Note that line4 defaults to *.

Messages

The COPY subcommand of EDIT causes error messages to be displayed at the terminal under specific conditions. To show these conditions, the following data set is assumed:

```
0010    A
0020    BB
0030    CCC
0040    DDDD
0050    EEEEE
0060    FFFFFF
0070    GGGGGGG
0080    HHHHHHHH
0090    IIIIIIIII
0100    JJJJJJJJJJ
0110    KKKKKKKKKKK
0120    LLLLLLLLLLLL
```

1. Entering

   copy * * *

   causes:

   INVALID OPERANDS * INVALID FOR COUNT OR END OF RANGE SPECIFICATION

2. Entering

   copy 10000 *

   causes:

   INVALID OPERANDS FIRST LINE TO BE MOVE/COPIED DOES NOT EXIST

3. Entering

   copy 'xyz' *

   causes:

   INVALID OPERANDS QUOTED STRING NOT FOUND

4. Entering

   copy 20 10 *

   causes:

   INVALID OPERANDS END OF RANGE MUST BE GREATER THAN OR EQUAL TO THE BEGINNING OF THE RANGE

5. Entering

   copy 20 '*' 100

   causes:

   INVALID OPERANDS STRING INVALID FOR END OF RANGE SPECIFICATION

6. Entering

   copy * 0 100

   causes:

   INVALID OPERANDS 0 INVALID FOR COUNT

7. Entering

   copy 10 40 20

   causes:

   INVALID OPERANDS TRYING TO MOVE/COPY INTO LINE RANGE

In the following examples, CLP refers to the current line pointer.

Example 1

Operation: Copy the current line right after itself in a line-numbered data set.

Known: Data set contains lines 10 through 120; current line pointer is at 50; EDIT provides an increment of 10.

Before:		***Enter:***					***After:***	
0010	A	copy	50	50	50		0010	A
0020	BB						0020	BB
0030	CCC	or					0030	CCC
0040	DDDD						0040	DDDD
0050	EEEEE	copy	50	50			0050	EEEEE
0060	FFFFFF					CLP	0060	EEEEE
0070	GGGGGGG	or					0061	FFFFFF
0080	HHHHHHHH						0070	GGGGGGG
0090	IIIIIIIII	copy	50				0080	HHHHHHHH
0100	JJJJJJJJJJ						0090	IIIIIIIII
0110	KKKKKKKKKK	or					0100	JJJJJJJJJJ
0120	LLLLLLLLLLL						0110	KKKKKKKKKK
		copy					0120	LLLLLLLLLLL
		or						
		copy	'ee'					

Example 2

Operation: Copy the current line right after itself in an unnumbered data set.

Known: Data set contains 12 lines of sequential alphabetic characters. Current line pointer is at the seventh line.

Before	***Enter:***					***After:***	
A		copy	*	1	*		A
BB							BB
CCC		or					CCC
DDDD							DDDD
EEEEE		copy	*	1			EEEEE
FFFFFF							FFFFFF
GGGGGGG		or					GGGGGGG
HHHHHHHH						CLP	GGGGGGG
IIIIIIIII		copy	*				HHHHHHHH
JJJJJJJJJJ							IIIIIIIII
KKKKKKKKKK		or					JJJJJJJJJJ
LLLLLLLLLLL							KKKKKKKKKK
		copy					LLLLLLLLLLL
		or					
		copy	'gg'				

Example 3

Operation: Illustrate an attempt to copy a line to a line before it.

Known: Data set contains lines 10 through 120; source line is 60; target line is 50; EDIT supplies increment of 10.

Before:

```
0010    A
0020    BB
0030    CCC
0040    DDDD
0050    EEEEE
0060    FFFFFF
0070    GGGGGGG
0080    HHHHHHHH
0090    IIIIIIIII
0100    JJJJJJJJJJ
0110    KKKKKKKKKK
0120    LLLLLLLLLLL
```

Enter:

```
copy    60  50
```

After:

```
        0010    A
        0020    BB
        0030    CCC
        0040    DDDD
        0050    EEEEE
CLP     0060    FFFFFF
        0061    FFFFFF
        0070    GGGGGGG
        0080    HHHHHHHH
        0090    IIIIIIIII
        0100    JJJJJJJJJJ
        0110    KKKKKKKKKK
        0120    LLLLLLLLLLL
```

Example 4

Operation: Find the line containing a specific word and copy it to the bottom of the data set.

Known: Data set contains nine lines of text; word to be found is "men"; data set is unnumbered.

Before:

```
NOW IS
THE TIME
FOR ALL
GOOD MEN
TO COME
TO THE
AID OF
THEIR
COUNTRY
```

Enter

```
top
copy    'men'   99999999
```

After:

```
        NOW IS
        THE TIME
        FOR ALL
        GOOD MEN
        TO COME
        TO THE
        AID OF
        THEIR
        COUNTRY
CLP     GOOD MEN
```

Example 5

Operation: Copy lines 10, 20, and 30 into a target area starting at line 100, using an increment of 5.

Known: Data set contains lines 10 through 120; EDIT provides increment of 10.

Before:

```
0010    A
0020    BB
0030    CCC
0040    DDDD
0050    EEEEE
0060    FFFFFF
0070    GGGGGGG
0080    HHHHHHHH
0090    IIIIIIIII
0100    JJJJJJJJJJ
0110    KKKKKKKKKK
0120    LLLLLLLLLLL
```

Enter:

```
copy    10  30  100 incr(5)
or
copy    9   31  100 incr(5)
or
copy    1   39  100 incr(5)
```

After:

```
        0010    A
        0020    BB
        0030    CCC
        0040    DDDD
        0050    EEEEE
        0060    FFFFFF
        0070    GGGGGGG
        0080    HHHHHHHH
        0090    IIIIIIIII
        0100    JJJJJJJJJJ
        0105    A
        0110    BB
CLP     0115    CCC
        0116    KKKKKKKKKK
        0120    LLLLLLLLLLL
```

Example 6

Operation: Copy four lines from a source area to a target area that overlaps the last line of the source, using the default increment.

Known: Data set contains lines 10 through 120; source lines are 20 through 50; target area starts at line 50; EDIT provides increment of 10.

Before:

```
0010    A
0020    BB
0030    CCC
0040    DDDD
0050    EEEEE
0060    FFFFFF
0070    GGGGGGG
0080    HHHHHHHH
0090    IIIIIIIII
0100    JJJJJJJJJJ
0100    KKKKKKKKKK
0120    LLLLLLLLLLL
```

Enter:

```
copy    20   50   50
```

After:

```
0010    A
0020    BB
0030    CCC
0040    DDDD
0050    EEEEE
0060    BB
0070    CCC
0080    DDDD
CLP 0090    EEEEE
0091    FFFFFF
0092    GGGGGGG
0093    HHHHHHHH
0094    IIIIIIIII
0100    JJJJJJJJJJ
0110    KKKKKKKKKK
0120    LLLLLLLLLLL
```

Example 7

Operation: Copy five lines into a target area that starts before but overlaps into the source area.

Known: Data set contains lines 10 through 120; source range is line 70 through line 110; target location is line 50; increment to be 10.

Before:

```
0010    A
0020    BB
0030    CCC
0040    DDDD
0050    EEEEE
0060    FFFFFF
0070    GGGGGGG
0080    HHHHHHHH
0090    IIIIIIIII
0100    JJJJJJJJJJ
0110    KKKKKKKKKK
0120    LLLLLLLLLLL
```

Enter:

```
copy    70   110  50   incr(10)
```

After:

```
0010    A
0020    BB
0030    CCC
0040    DDDD
0050    EEEEE
0060    GGGGGGG
0070    HHHHHHHH
0080    IIIIIIIII
0090    JJJJJJJJJJ
CLP 0100    KKKKKKKKKK
0101    FFFFFF
0102    GGGGGGG
0103    HHHHHHHH
0104    IIIIIIIII
0105    JJJJJJJJJJ
0110    KKKKKKKKKK
0120    LLLLLLLLLLL
```

Example 8

Operation: Copy three lines to the top of the data set at line 0.

Known: Data set contains lines 10 through 120; line 0 does not exist; source lines are 80, 90, and 100; target area starts at line 0.

Before:

```
0010    A
0020    BB
0030    CCC
0040    DDDD
0050    EEEEE
0060    FFFFFF
0070    GGGGGGG
0080    HHHHHHHH
0090    IIIIIIIII
0100    JJJJJJJJJJ
0110    KKKKKKKKKK
0120    LLLLLLLLLLL
```

Enter:

```
top
copy      80   100 *    incr(50)
or
copy      80   100 0    incr(50)
```

After:

```
0000    HHHHHHHH
0050    IIIIIIIII
CLP 0100    JJJJJJJJJJ
0101    A
0102    BB
0103    CCC
0104    DDDD
0105    EEEEE
0106    FFFFFF
0107    GGGGGGG
0108    HHHHHHHH
0109    IIIIIIIII
0110    JJJJJJJJJJ
0111    KKKKKKKKKK
0120    LLLLLLLLLLL
```

Example 9

Operation: Copy three lines to the top of the data set at line 0, using an increment of 50.

Known: Data set contains lines 0 through 120; line 0 contains data; source lines are 80, 90, and 100; target area starts at line 0.

Before:

```
0000    ZIP
0010    A
0020    BB
0030    CCC
0040    DDDD
0050    EEEEE
0060    FFFFFF
0070    GGGGGGG
0080    HHHHHHHH
0090    IIIIIIIII
0100    JJJJJJJJJJ
0110    KKKKKKKKKK
0120    LLLLLLLLLLL
```

Enter:

```
top
copy      80   100 *    incr(50)
```

The attempt to copy into line 0 gets the target data to the top of the data set but shifts the target line by the increment value

After:

```
0050    HHHHHHHH
0100    IIIIIIIII
CLP 0150    JJJJJJJJJJ
0151    ZIP
0152    A
0153    BB
0154    CCC
0155    DDDD
0156    EEEEE
0157    FFFFFF
0158    GGGGGGG
0159    HHHHHHHH
0160    IIIIIIIII
0161    JJJJJJJJJJ
0162    KKKKKKKKKK
0163    LLLLLLLLLLL
```

Note: An entry of

```
copy      80   100 0    incr(50)
```

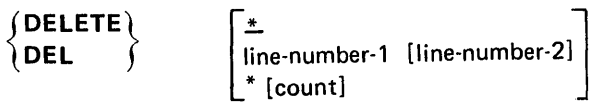
produces the results shown at right. The target data is inserted between line 0 and the remainder of the data set.

```
0000    ZIP
0050    HHHHHHHH
0100    IIIIIIIII
CLP 0150    JJJJJJJJJJ
0151    A
0152    BB
0153    CCC
0154    DDDD
0155    EEEEE
0156    FFFFFF
0157    GGGGGGG
0158    HHHHHHHH
0159    IIIIIIIII
0160    JJJJJJJJJJ
0161    KKKKKKKKKK
0162    LLLLLLLLLLL
```

DELETE Subcommand of EDIT

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.



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
    del *
or
    del
or
    *
```

Any of the preceeding 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: 00004

```
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
```

DOWN Subcommand of EDIT

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.

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 the EDIT command. This subcommand may be used with or without the optional keywords SAVE or NOSAVE. In either case, the EDIT command terminates processing. If you have modified your data set and have not entered the SAVE subcommand or the SAVE/NOSAVE operand on END, the system will ask you if you want to save the data set. At this point, you may reply SAVE if you wish to save the data set. If you do not wish to save the data set, reply END.

END [SAVE / NOSAVE]

Note: There are no defaults. If a keyword is not specified, and SAVE was not entered after the last modification, the user will be prompted by the system.

Regardless of the user's PROMPT/NOPROMPT option, when END (with no operands) is found in a CLIST, edit-mode is terminated. (There is no SAVE processing done for this portion of the session.) If END (with no operands) is found outside a CLIST, the user is prompted to enter END or SAVE regardless of the PROMPT/NOPROMPT option.

SAVE
specifies that the modified data set is to be saved.

NOSAVE
specifies that the modified data set is not to be saved.

END Subcommand of EDIT

Use the END subcommand to terminate the EDIT command. This subcommand may be used with or without the optional keywords SAVE or NOSAVE. In either case, the EDIT command terminates processing. If you have modified your data set and have not entered the SAVE subcommand or the SAVE/NOSAVE operand on END, the system will ask you if you want to save the data set. At this point, you may reply SAVE if you wish to save the data set. If you do not wish to save the data set, reply END.

END [SAVE / NOSAVE]

Note: There are no defaults. If a keyword is not specified, and SAVE was not entered after the last modification, the user will be prompted by the system.

SAVE
specifies that the modified data set is to be saved.

NOSAVE
specifies that the modified data set is not to be saved.

EXEC Subcommand of EDIT

Use the EXEC subcommand to execute a command procedure. Refer to the EXEC command for the description of the syntax and function of the EXEC subcommand.

FIND Subcommand of EDIT

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.

{FIND / F} [string [position]]

Note: 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. If you issue the TOP subcommand, the search for the specified string begins with the second line of the data set. Successive use of the FIND subcommand without operands allows you to search a data set, line by line.

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, for example, 'string character'.
2. A single quote within a character string is represented by two single quotes, for example, 'pilgrims''s progress'.
3. A null string is represented by two single quotes, for example, ''.

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. For COBOL data sets, the starting column is calculated from the end of the six-digit line number. (If you want to consider a string starting in column 8, you must specify the digit 2 for this 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
```

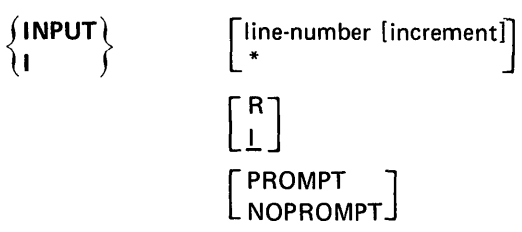
HELP Subcommand of EDIT

Use the HELP subcommand to obtain the syntax and function of EDIT subcommands.

Refer to the HELP command for a description of the syntax and function of the HELP subcommand.

INPUT Subcommand of EDIT

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.



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. If the increment is greater than 1, all lines between the replacement lines will be deleted.

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:

- The value (either PROMPT or NOPROMPT) that was established the last time you used input mode

- PROMPT, if this is the first use of input mode and the data set has line numbers
- 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 display at your terminal will resemble the following with your input in lowercase and the system's response in uppercase.

```
edit quer 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.
The display at your terminal will resemble the following:

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

INSERT Subcommand of EDIT

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, and FIND subcommands.

{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:

```
"UCBLFG DS AL1 CONTROL FLAGS"
```

The data set is not line-numbered.
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 display at your terminal will be similar to the following:

```
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 40-007830 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.

The display at your terminal will be similar to the following:

```
down 24
insert
INPUT
00281 j.w.house 13-244831 24.73
00282 t.n.howard 24-782095 3.05
00283 b.h.ireland 40-007830 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.

Insert/Replace/Delete Function of EDIT

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.

{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.

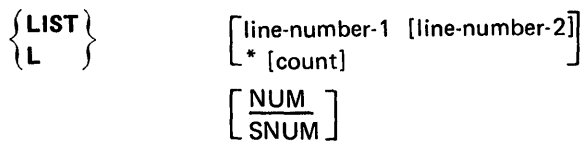
```
98
```

or

```
*
```

LIST Subcommand of EDIT

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



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 current line pointer 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.

Note: If the current line pointer is at zero (for example, as a result of a TOP command), and line zero is not already in the data set, the current line pointer moves to the first existing line.

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, that is, not displayed at the terminal.

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
```

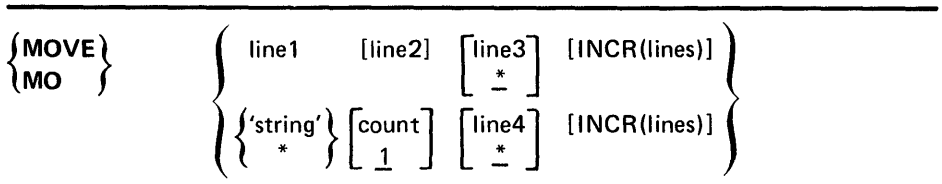
MOVE Subcommand of EDIT

Use the MOVE subcommand of EDIT to move one or more records that exist in the data set being edited. The move operation moves data from a source location to a target location within the same data set and deletes the source data. Existing lines in the target area are shifted toward the end of the data set as required to make room for the incoming data. No lines are lost in the shift.

The target line cannot be within the source area, with the exception that the target line (the first line of the target area) can overlap the last line of the source area.

Upon completion of the move operation, the current line pointer points to the last moved-to line, not to the last line shifted to make room in the target area.

Note: If you cause an attention interruption during the move operation, the data set may be only partially changed. As a check, list the affected part of the data set before continuing.



Note: MOVE without operands is ignored.

line1
specifies the first line or the lower limit of the range to be moved. If the specified line number does not exist in this data set, the range begins the next higher line number.

line2
specifies the last line or the upper limit of the range to be moved. If the specified line number does not exist in this data set, the range ends with the highest line number that is less than line2. If line2 is not entered, the value defaults to the value of line1; that is, the source becomes one line. Do not enter asterisk for line2.

Note: If MOVE is followed by two line-number operands, the system assumes them to represent line1 and line3, and defaults line2 to the value of line1.

line3
specifies the target line number; that is, the line at which the moved-to data area will start. If the line3 value corresponds to an existing line, the target line is changed to line3 + INCR(lines) and either becomes a new line or displaces an existing line at that location. Once the move operation begins, existing lines encountered in the target area are renumbered to make room for the incoming data. The increment for renumbered lines is one (1). Specifying zero (0) for line3 puts the moved data at the top of the data set only if line 0 is empty; if line 0 has data, enter TOP followed by MOVE with line3 set to *. Note that line3 defaults to *.

Note: The value of line3 should not fall in the range from line1 to line2; that is, the target line must not be in the range being moved. Exception: line3 can be equal to line2.

*****
represents the value of the current line pointer.

INCR(lines)
specifies the line number increment to be used for this move operation. The default is the value in effect for this data before the move operation. When the move operation is complete, the increment reverts to the value in effect before MOVE was issued. Range: 1-8 decimal digits but not zero.

Note: The increment for any renumbered line is one (1).

'string'
specifies a string of alphameric characters with a maximum length equal to or less than the logical record length of the data set being edited. When a character string is specified, a search starting at the current line is done for the line containing the string. When found, that line is the start of the range to be moved for either numbered or unnumbered data sets.

count
specifies the total number of lines (the range) to be moved. The default for count is one (1). Enter 1-8 decimal digits but not zero (0) or asterisk (*).

line4
applies to both numbered and unnumbered data sets. For unnumbered data sets, line4 specifies the target line (the line at which the moved-to data area will start) as a relative line number (the nth line in the data set). For numbered data sets, line4 is specified the same as line3. Specifying zero (0) for line4 puts the moved data at the top of the data set only if line 0 is empty; if line 0 has data, enter TOP followed by MOVE with line4 set to *. Note that line4 defaults to *.

Messages

The MOVE subcommand of EDIT causes error messages to be displayed at the terminal under specific conditions. To show these conditions, the following data set is assumed:

```
0010    A
0020    BB
0030    CCC
0040    DDDD
0050    EEEEE
0060    FFFFFF
0070    GGGGGGG
0080    HHHHHHHH
0090    IIIIIIIII
0100    JJJJJJJJJJ
0110    KKKKKKKKKK
0120    LLLLLLLLLLL
```

1. Entering
   move * * *
   causes:
   IKJ52579I INVALID OPERANDS * INVALID FOR COUNT OR END OF RANGE SPECIFICATION
2. Entering
   move 100000 *
   causes:
   IKJ52579I INVALID OPERANDS FIRST LINE TO BE MOVE/COPIED DOES NOT EXIST
3. Entering
   move 'xyz' *
   causes:
   IKJ52579I INVALID OPERANDS QUOTED STRING NOT FOUND
4. Entering
   move 20 to 10 *
   causes:
   IKJ52579I INVALID OPERANDS END OF RANGE MUST BE GREATER THAN OR EQUAL TO THE BEGINNING OF THE RANGE
5. Entering
   move 20 '*' 100
   causes:
   IKJ52579I INVALID OPERANDS STRING INVALID FOR END OF RANGE SPECIFICATION
6. Entering
   move * 0 100
   causes:
   IKJ52579I INVALID OPERANDS 0 INVALID FOR COUNT
7. Entering
   move 10 40 20
   causes:
   IKJ52579I INVALID OPERANDS TRYING TO MOVE/COPY INTO LINE RANGE

In the following examples, CLP refers to the current line pointer.

Example 1

Operation: Move the current line right after itself in a line-numbered data set.

Known: Data set contains lines 10 through 120; current line pointer is at 50; EDIT provides an increment of 10.

Before:	*Enter:*	*After:*
0010 A	move 50 50 50	0010 A
0020 BB		0020 BB
0030 CCC	or	0030 CCC
0040 DDDD		0040 DDDD
0050 EEEEE	move 50 50	CLP 0060 EEEEE
0060 FFFFFF		0061 FFFFFF
0070 GGGGGGG	or	0070 GGGGGGG
0080 HHHHHHHH		0080 HHHHHHHH
0090 IIIIIIIII	move 50	0090 IIIIIIIII
0100 JJJJJJJJJJ		0100 JJJJJJJJJJ
0110 KKKKKKKKKK	or	0110 KKKKKKKKKK
0120 LLLLLLLLLLL		0120 LLLLLLLLLLL
	move 'ee'	

Note: MOVE is ignored without operands.

Example 2

Operation: Move the current line right after itself in an unnumbered data set.

Known: Data set contains 12 lines of sequential alphabetic characters. Current line pointer is at the seventh line.

Before	*Enter:*	*After:*
A	move * 1 *	A
BB		BB
CCC	or	CCC
DDDD		DDDD
EEEEE	move * 1	EEEEE
FFFFFF		FFFFFF
GGGGGGG	or	CLP GGGGGGG
HHHHHHHH		HHHHHHHH
IIIIIIIII	move *	IIIIIIIII
JJJJJJJJJJ		JJJJJJJJJJ
KKKKKKKKKK	or	KKKKKKKKKK
LLLLLLLLLLL		LLLLLLLLLLL
	move 'gg'	

Note: The effect of the operation is an unchanged data set.

Example 3

Operation: Illustrate an attempt to move a line to a line before it.

Known: Data set contains lines 10 through 120; source line is 60; target line is 50; EDIT supplies increment of 10.

Before:

```
0010 A
0020 BB
0030 CCC
0040 DDDD
0050 EEEEE
0060 FFFFFF
0070 GGGGGGG
0080 HHHHHHHH
0090 IIIIIIIII
0100 JJJJJJJJJJ
0110 KKKKKKKKKK
0120 LLLLLLLLLLL
```

Enter:

```
move 60 50
```

After:

```
     0010 A
     0020 BB
     0030 CCC
     0040 DDDD
     0050 EEEEE
CLP  0060 FFFFFF
     0070 GGGGGGG
     0080 HHHHHHHH
     0090 IIIIIIIII
     0100 JJJJJJJJJJ
     0110 KKKKKKKKKK
     0120 LLLLLLLLLLL
```

Example 4

Operation: Find the line containing a specific word and move it to the bottom of the data set.

Known: Data set contains nine lines of text; word to be found is "men"; data set is unnumbered.

Before:

```
NOW IS
THE TIME
FOR ALL
GOOD MEN
TO COME
TO THE
AID OF
THEIR
COUNTRY
```

Enter

```
top
move 'men' 99999999
```

After:

```
     NOW IS
     THE TIME
     FOR ALL
     TO COME
     TO THE
     AID OF
     THEIR
     COUNTRY
CLP  GOOD MEN
```

Example 5

Operation: Move lines 10, 20, and 30 into a target area starting at line 100, using an increment of 5.

Known: Data set contains line 10 through 120; EDIT provides increment of 10.

Before:

```
0010 A
0020 BB
0030 CCC
0040 DDDD
0050 EEEEE
0060 FFFFFF
0070 GGGGGGG
0080 HHHHHHHH
0090 IIIIIIIII
0100 JJJJJJJJJJ
0110 KKKKKKKKKK
0120 LLLLLLLLLLL
```

Enter:

```
move 10 30 100 incr(5)

or

move 9 31 100 incr(5)

or

move 1 39 100 incr(5)
```

After:

```
     0040 DDDD
     0050 EEEEE
     0060 FFFFFF
     0070 GGGGGGG
     0080 HHHHHHHH
     0090 IIIIIIIII
     0100 JJJJJJJJJJ
     0105 A
     0110 BB
CLP  0115 CCC
     0116 KKKKKKKKKK
     0120 LLLLLLLLLLL
```

Example 6

Operation: Move four lines from a source area to a target area that overlaps the last line of the source, using the default increment.

Known: Data set contains lines 10 through 120; source lines are 20 through 50; target area starts at line 50; EDIT provides increment of 10.

Before:

```
0010 A
0020 BB
0030 CCC
0040 DDDD
0050 EEEEE
0060 FFFFFF
0070 GGGGGGG
0080 HHHHHHHH
0090 IIIIIIIII
0100 JJJJJJJJJJ
0110 KKKKKKKKKK
0120 LLLLLLLLLLL
```

Enter:

```
move 20 50 50
```

After:

```
     0010 A
     0060 BB
     0070 CCC
     0080 DDDD
CLP  0090 EEEEE
     0091 FFFFFF
     0092 GGGGGGG
     0093 HHHHHHHH
     0094 IIIIIIIII
     0100 JJJJJJJJJJ
     0110 KKKKKKKKKK
     0120 LLLLLLLLLLL
```

Example 7

Operation: Move five lines into a target area that starts before but overlaps into the source area.

Known: Data set contains lines 10 through 120; source range is line 70 through line 110; target location is line 50; increment to be 10.

Before

```
0010 A
0020 BB
0030 CCC
0040 DDDD
0050 EEEEE
0060 FFFFFF
0070 GGGGGGG
0080 HHHHHHHH
0090 IIIIIIIII
0100 JJJJJJJJJJ
0110 KKKKKKKKKK
0120 LLLLLLLLLLL
```

Enter:

```
move 70 110 50 incr(10)
```

After:

```
     0010 A
     0020 BB
     0030 CCC
     0040 DDDD
     0050 EEEEE
     0060 GGGGGGG
     0070 HHHHHHHH
     0080 IIIIIIIII
     0090 JJJJJJJJJJ
CLP  0100 KKKKKKKKKK
     0101 FFFFFF
     0120 LLLLLLLLLLL
```

Example 8

Operation: Move three lines to the top of the data set at line 0.

Known: Data set contains lines 10 through 120; line 0 doesn't exist; source lines are 80, 90, and 100; target area starts at line 0.

Before:

```
0010 A
0020 BB
0030 CCC
0040 DDDD
0050 EEEEE
0060 FFFFFF
0070 GGGGGGG
0080 HHHHHHHH
0090 IIIIIIIII
0100 JJJJJJJJJJ
0110 KKKKKKKKKK
0120 LLLLLLLLLLL
```

Enter

```
top
move 80 100 * incr(50)
                      CLP
or

move 80 100 0 incr(50)
```

After:

```
0000 HHHHHHHH
0050 IIIIIIIII
0100 JJJJJJJJJJ
0101 A
0102 BB
0103 CCC
0104 DDDD
0105 EEEEE
0106 FFFFFF
0107 GGGGGGG
0110 KKKKKKKKKK
0120 LLLLLLLLLLL
```

Example 9

Operation: Move three lines to the top of the data set at line 0, using an increment of 50.

Known: Data set contains lines 0 through 120; line 0 contains data; source lines are 80, 90, and 100; target area starts at line 0.

Before:

```
0000 ZIP
0010 A
0020 BB
0030 CCC
0040 DDDD
0050 EEEEE
0060 FFFFFF
0070 GGGGGGG
0080 HHHHHHHH
0090 IIIIIIIII
0100 JJJJJJJJJJ
0110 KKKKKKKKKK
0120 LLLLLLLLLLL
```

Enter:

```
top
move 80 100 * incr(50)
                      CLP
The attempt to move into
line 0 gets the target data
to the top of the data set
but shifts the target line
by the increment value.
```

Note: An entry of

```
move 80 100 0 incr(50)
```

produces the results shown at right. The target data is inserted between line 0 and the remainder of the data set.

After:

```
0050 HHHHHHHH
0100 IIIIIIIII
0150 JJJJJJJJJJ
0151 ZIP
0152 A
0153 BB
0154 CCC
0155 DDDD
0156 EEEEE
0157 FFFFFF
0158 GGGGGGG
0159 KKKKKKKKKK
0160 LLLLLLLLLLL
```

After:

```
0000 ZIP
0050 HHHHHHHH
0100 IIIIIIIII
0150 JJJJJJJJJJ
0151 A
0152 BB
0153 CCC
0154 DDDD
0155 EEEEE
0156 FFFFFF
0157 GGGGGGG
0158 KKKKKKKKKK
0159 LLLLLLLLLLL
```

PROFILE Subcommand of EDIT

Use the PROFILE subcommand to change the characteristics of your user profile. Refer to PROFILE command for a discussion of the syntax and function of PROFILE subcommand.

RENUM Subcommand of EDIT

Use the RENUM subcommand to:

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

New line numbers are placed in the last eight character positions if the data set being edited contains fixed-length records. There are three exceptions to this general rule:

- Data set type COBOL - first six positions
- Data set type VSBASIC - first five positions
- Data set type ASM and NUM keyword specified on EDIT command - positions indicated in NUM keyword subfield.

If fixed-length record data sets are being numbered for the first time, any data in the positions indicated above is overlaid.

If variable-length records without sequence numbers are being edited, the records will be lengthened so that an eight-digit sequence field (five-digits if VSBASIC) is prefixed to each record. You are notified if any records have been truncated in the process. (Records are truncated when the data length plus the sequence length exceeds the maximum record length of the data set being edited.)

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

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

---

new-line-number
specifies the new line number to be assigned to the first line renumbered. 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.

end-line-number
specifies the line number at which renumbering is to end. If this operand is omitted, renumbering continues to the end of the data set. You cannot use this operand without specifying all the other operands.

Example 1

Operation: Renumber an entire data set using the default values for each operand.

```
renum
```

Example 2

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.

Example 4

Operation: Renumber a range of lines so that new lines may be inserted.

Known:
Before renumbering, the data set lines are numbered 10,20,23,26,29,30,40, and 50.
Two lines are to be inserted after line 29.
Lines 23-29 are to be renumbered with an increment of 2.
The first new number to be assigned is 22.

```
ren 22 2 23 29
```

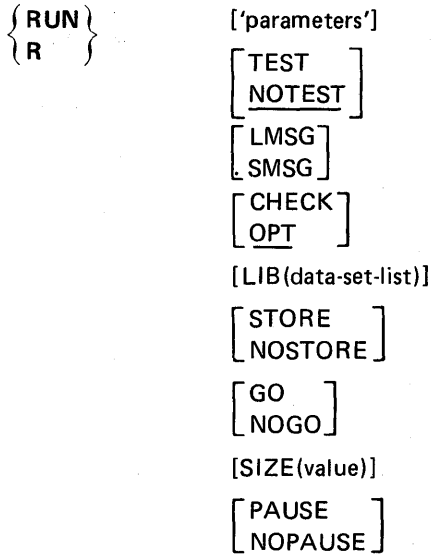
Use the RUN subcommand to compile, load, and execute the source statements in the data set that you are editing. The RUN subcommand is designed specifically for use with certain program products; it selects and invokes the particular program product needed to process your source statements. Figure 8 shows which program product is selected to process each type of source statement.

Notes:

1. Any data sets required by your problem program may be allocated before you enter EDIT mode or may be allocated using the ALLOCATE subcommand.
2. If you wish to enter a value for 'parameters,' you should enter this prior to any of the other keyword operands.

If your program or data set contains statements of this type (see EDIT):	Then the following Program Product (or equivalent) can be used:
ASM	TSO ASM Prompter
COBOL	TSO COBOL Prompter and OS Full American National Standard COBOL Version 3 or Version 4
FORTGI	TSO FORTRAN Prompter and FORTRAN IV (G1)
GOFORT	Code and Go FORTRAN
PLI	PL/I Checkout Compiler or PL/I Optimizing Compiler
VSBASIC	VSBASIC
You can use the CONVERT command to convert Code and Go FORTRAN free-form statements to a form suitable for the FORTRAN compiler. 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 set types.	

Figure 8. Source Statement/Program Product Relationship



'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.

TEST
specifies that testing will be performed during execution. This operand is valid for the VSBASIC program product 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 Code and Go FORTRAN program product only.

SMSG
specifies that you want to receive the short, concise diagnostic messages. This is the default for Code and Go FORTRAN program product.

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 for data sets having the PLI descriptive qualifier.

OPT
specifies the PL/I Optimizing compiler. This operand is valid for the PL/I program product only. This is the default value for data sets having the PLI descriptive qualifier if both CHECK and OPT are omitted.

LIB(data-set-list)
specifies the library or libraries that contain subroutines needed by the program you are running. These libraries are concatenated to the default system libraries and passed to the loader for resolution of external references. This operand is valid only for the following data set types: ASM, COBOL, FORTGI, and PLI(Optimizer).

STORE
specifies that a permanent OBJ data set is to be created. The dsname of the OBJ data set is based on the data set name entered on the EDIT command. This operand is valid only for VSBASIC statements.

NOSTORE
specifies that a permanent OBJ data set is not to be created. This operand is valid only for VSBASIC statements.

GO
specifies that the compiled program is to be executed. This operand is valid only for VSBASIC statements.

NOGO
specifies that the compiled program is not to be executed. This operand is valid only for VSBASIC statements.

SIZE(value)
specifies the size (1-999) of the user area for VSBASIC.

PAUSE
specifies that the user is to be given the chance to add or change certain compiler options before proceeding to the next chain program. This operand is valid only for VSBASIC statements.

NOPAUSE
specifies that the user is not to be given the chance to add or change certain compiler options before proceeding to the next chain program. This operand is valid only for VSBASIC statements.

Example 1

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'
```

Example 2

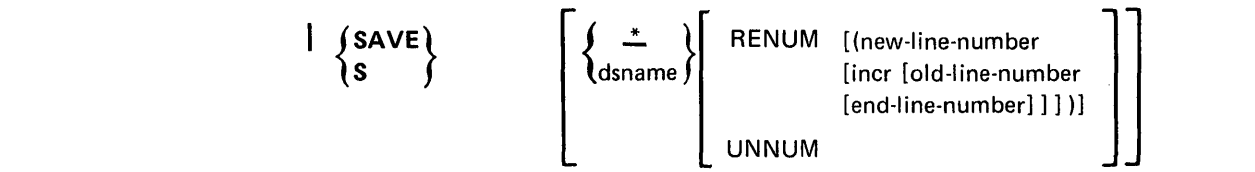
Operation: Run a FORTRAN IV (GI) program that calls an assembler language output program to manipulate bit patterns.

Known:
The assembler language subroutine in load module form resides in a library called USERID.MYLIB.LOAD.

```
run lib(mylib.load)
```

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.



*

specifies that the edited version of your data set is to replace the original version. This is the default, if there are no operands entered on the subcommand.

dsname

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 or an asterisk 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.

Note: The following operands cannot be included unless data set name or an * is specified.

RENUM

specifies that the data set will be renumbered before it is saved.

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.

incr
specifies the amount by which each succeeding line number is to be incremented. The default is 10. This operand cannot be included unless the new-line-number is specified.

old-line-number
specifies the line location within the data set where the renumber process will begin. If this operand is omitted, renumbering will start at the beginning of the data set. The old-line-number must be equal to or less than the new-line-number. This operand cannot be included unless "incr" is specified.

end-line-number
specifies the line location within the data set where renumbering is to end. If this operand is omitted, renumbering stops at the end of the data set. The end-line-number must be greater than the old-line-number. This operand cannot be included unless the old-line-number is specified.

UNNUM
specifies that the data set will be unnumbered before it is saved.

Note: If the data set being edited originally contained control characters (ASCII or machine), and you enter SAVE without operands, the following actions apply.

Sequential Data Set

- You will be warned that the data set will be saved without control characters, that is, the record format will be changed.
- You will be prompted to enter another data set name for SAVE or a null line to reuse the EDIT data set.

Partitioned Data Set

Saving into the EDIT data set is not allowed when it is partitioned with a control character attribute. You must save into another data set by specifying a data-set-name on a subsequent SAVE subcommand entry.

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
```

Example 2

Operation: Save the data set that has just been edited, renumbering it first.

Known:

new-line-number	100
increment(INCR)	50

```
save * renum( 100 50 )
```

SCAN Subcommand of EDIT

Use the SCAN subcommand to request syntax checking services for statements that will be processed by the FORTRAN(H) compiler or by the Code and Go FORTRAN, or FORTRAN IV (G1), program products. You can have each statement checked as you enter it in input mode, or any or all existing statements checked. You must explicitly request a check of the syntax of statements you are adding, replacing, or modifying, via the CHANGE subcommand, the INSERT subcommand with the insert-data operand, or the insert/replace/delete function.

```
{SCAN}   [line-number-1 [line-number-2]]
{SC  }   [* [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
```

SEND Subcommand of EDIT

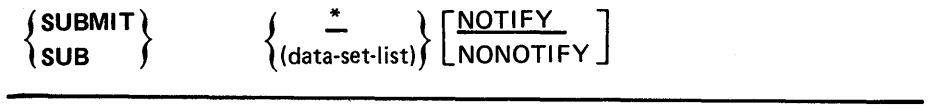
Use the SEND subcommand to send a message to another terminal user or to the system operator. Refer to the SEND command for a description of the syntax and function of the SEND subcommand.

SUBMIT Subcommand of EDIT

Use the SUBMIT subcommand of EDIT to submit one or more batch jobs for conventional processing. Each job submitted must reside in either a sequential data set, a direct-access data set or in a member of a partitioned data set. Submitted data sets must be fixed blocked, 80 byte records. Using the EDIT command to create a CNTL data set will provide the correct format.

Any of these data sets can contain part of a job, one job, or more than one job that can be executed via a single entry of SUBMIT. Each job must comprise an input job stream (JCL plus data). Do not submit data sets with descriptive qualifiers TEXT or PLI if the characters in these data sets are lower case.

Job cards are optional. The generated jobname will be your userid plus an identifying character. SUBMIT will prompt you for the character and insert the job accounting information from the user's LOGON command on any generated job card. The system or installation default MSGCLASS and CLASS are used for submitted jobs unless MSGCLASS and CLASS are specified on the job card(s) being submitted. See the first section in Appendix A for an example of a generated JOB card.



*****

specifies that the data set being edited defines the input stream to be submitted. This is the default if no operands are entered on the subcommand.

data-set-list

specifies one or more data set names or names of members of partitioned data sets that define an input stream (JCL plus data). If you specify more than one data set name, enclose them in parentheses.

Note: Either an asterisk or the data-set-list must be specified if any keywords are used.

NOTIFY

specifies that you are to be notified when your job terminates in the background if a JOB statement has not been provided. 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. Notify is the default value if a JOB statement is generated.
If you supply your own JOB statement, use the NOTIFY=userid keyword on the JOB statement if you wish to be notified when the job terminates. SUBMIT ignores the NOTIFY keyword unless it is generating a JOB statement.

NONOTIFY
specifies that a termination message will not be issued or placed in the broadcast data set. The NONOTIFY keyword is only recognized when a JOB statement has not been provided with the job that you are processing. If you supply your own JOB statement, you must use the NOTIFY= userid keyword on the JOB statement to receive notification.

Notes:

- If any of the above types of data sets containing two or more jobs is submitted for processing, certain conditions apply.
  The SUBMIT processor will build a job card for the first job in the first data set, if none was supplied, but will not build job cards for any other jobs in the data set(s).
  If the SUBMIT processor determines that the first job contains an error, none of the jobs are submitted.
  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.
- Any job card you supply should have a job name consisting of your userid and a single identifying character. If the jobname is not in this format, you will not be able to refer to it with the CANCEL command. You will be required to specify the jobname in the STATUS command if the IBM-supplied exit has not been replaced by your installation and your job name is not your userid plus a single identifying character.
- If you wish to provide a job card but you also want to be prompted for a unique jobname character, put your userid in the jobname field and follow it with blanks so that there is room for SUBMIT to insert the prompted-for character. This allows you to change jobnames without re-editing the JCL data set.
- Once SUBMIT has successfully submitted a job for conventional batch processing, it will issue a 'jobname(jobid) submitted' message. The jobid is a unique job identifier assigned by the job entry subsystem.
- This subcommand may be used only by personnel who have been given the authority to do so by the installation management.

Example

Operation: Submit the data set being edited for batch processing.

Known:
The data set has no job card and you do not want to be notified when the job is completed.

```
submit  * nonotify
```

TABSET Subcommand of EDIT

Use the TABSET subcommand to:

- Establish or change the logical tabulation settings.
- Cancel 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 9 will be in effect when you first enter the EDIT command. (See *TSO Terminal User's Guide* to determine if your terminal supports tab setting.)

Data Set Name Descriptive Qualifier	Default Tab Settings Columns
ASM	10,16,31,72
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(H) compilers, FORTRAN IV (G1) and Code and Go FORTRAN program product data set types.	7,72
PLI PL/I Checkout and Optimizing compiler data set types.	5,10,15,20,25,30,35,40,45,50
TEXT	5,10,15,20,30,40
VSBASIC	10,15,20,25,30,35,40,45,50,55
User-defined	10,20,30,40,50,60

Figure 9. Default Tab Settings

You may find it convenient to have the mechanical tab settings coincide with the logical tab settings. Note that, except for line-numbered COBOL or VSBASIC 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 and VSBASIC 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 or VSBASIC 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; (for COBOL) or the first five columns (for VSBASIC); 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 six columns being occupied by the system-supplied sequence number (line number). In line-numbered VSBASIC data sets, the column following a line number

prompt is considered to be column 6 of your data, the first five columns being occupied by the system-supplied sequence number.

```
{TABSET}     [ON [(integer-list)]]
{TAB   }     [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". Ten 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
123456789t123456789t123...
```

TOP Subcommand of EDIT

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.

TOP

Example 1

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

Known:
The data set is not line-numbered.

```
top
```

UNNUM Subcommand of EDIT

Use the UNNUM subcommand to remove existing line numbers from the records in the data set.

{UNNUM}
{UNN }

Example 1

Operation: Remove the line numbers from an ASM-type data set.

Known:
The data set has line numbers.

```
unnum
```

UP Subcommand of EDIT

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.

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
```

VERIFY Subcommand of EDIT

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.

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.

Note: Subcommands that change the current line pointer and cause it to be displayed if the VERIFY subcommand is activated are BOTTOM, CHANGE, COPY, DELETE, DOWN, FIND, MOVE, RENUM, UNNUM and UP.

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
```

END Command

You may use the END command to end a command procedure. When the system encounters an END command in a command procedure, execution of the command procedure is halted. This function is better performed by the EXIT statement.

END

EXEC Command

Use the EXEC command to execute a command procedure.

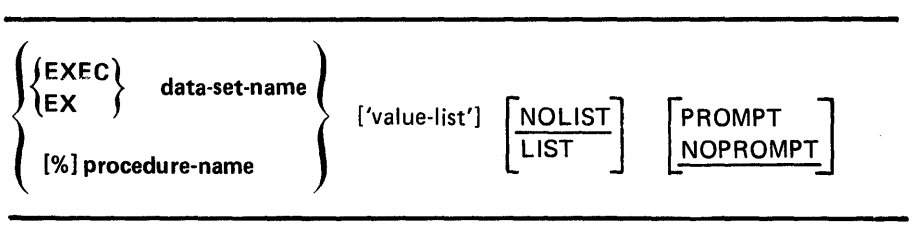
You can specify the EXEC command or the EXEC subcommand of EDIT in three ways:

- **The explicit form:** Enter EXEC or EX followed by the name of the data set that contains the command procedure.
- **The implicit form:** Do *not* enter EXEC or EX; only enter the procedure-name (a member of a command procedure library). A command procedure library is a partitioned data set that must be allocated to the SYSPROC file name either dynamically by the ALLOCATE command or as part of the LOGON procedure. TSO will determine if the name is a system command before searching SYSPROC for the procedure.
- **The extended implicit form:** Enter a percent sign followed by the procedure-name. TSO will only search the SYSPROC file for the specified name. For procedures that reside in SYSPROC, this form is the faster of the implicit forms.

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

Note: For more information concerning symbolic variables and command procedures, refer to the section "Command Procedures" in this book. Command procedures are explained in greater detail in *OS/VS2 TSO Terminal User's Guide*.

The EXEC command and the EXEC subcommand of EDIT perform the same basic functions. However, a command procedure which is executed with the EXEC subcommand of EDIT can only execute command procedure statements and EDIT subcommands.



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, you must enclose the fully-qualified name within apostrophes and the data set must contain line numbers according to the following format:

```
Variable blocked - First eight characters in each record
Fixed blocked - Last eight characters in each record
```

Since any data contained in these columns is lost, you should not enter data in these columns.

[%]procedure-name
specifies a member of a command procedure library. If the percent sign (%) is entered, TSO will search only the SYSPROC file for the specified name.

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 to replace the *positional operands* in the PROC statement must be in the same sequence as the positional operands. The actual values to replace the *keywords* in the PROC statement must follow the positional values, but may be in any sequence. A keyword defined on the PROC statement may have a value consisting of a character string with delimiters, provided that the character string is enclosed in quotes. 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. If a quoted string appears as the value of a keyword within the value list, the number of quotes must be doubled again (see example 3).

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.

PROMPT
specifies that prompting to the terminal will be allowed during the execution of a command procedure. The PROMPT keyword implies LIST, unless NOLIST has been explicitly specified. Therefore, all commands and subcommands will be listed at the terminal as they are executed. This operand is valid only for the explicit form of EXEC.

NOPROMPT
specifies no prompting during the execution of a command procedure. This is the default if neither PROMPT nor NOPROMPT is specified.

Notes:

1. The PROMPT keyword is not propagated to nested EXEC commands. PROMPT must be specified on a nested EXEC command if you wish to be prompted during execution of the command procedure it invokes.
2. No prompting will be allowed during the execution of a command procedure if the NOPROMPT keyword operand of PROFILE has been specified, even if the PROMPT option of EXEC has been specified.
3. The following is a list of options resulting from specific keyword entries:

Keyword specified	Resulting options
PROMPT	PROMPT LIST
NOPROMPT	NOPROMPT NOLIST
LIST	LIST NOPROMPT
NOLIST	NOLIST NOPROMPT
PROMPT LIST	PROMPT LIST
PROMPT NOLIST	PROMPT NOLIST
NOPROMPT LIST	NOPROMPT LIST
NOPROMPT NOLIST	NOPROMPT NOLIST
No keywords	NOPROMPT NOLIST

Suppose the following command procedure exists as a data set named ANZAL:

```
proc 3 input output list lines( )
allocate dataset(&input) file(indata) old
allocate dataset(&output) block(100) space(300,100)
allocate dataset(&list) file(print)
call proc2 '&lines'
end
```

Note: If the symbolic value must be immediately followed by a period, the symbolic value must end with a period. (A single period following a symbolic value is ignored.)

The PROC statement indicates that the three symbolic values, & INPUT, & OUTPUT and & LIST, are positional (required) and that the symbolic value & LINES is a keyword (optional).

To replace ALPHA for INPUT, BETA for OUTPUT, COMMENT for LIST and 20 for LINES, you would enter: (implicit form)

```
anzal alpha beta comment lines(20)
```

Example 1

Operation: Execute a command procedure to invoke the assembler.

Known: The name of the data set that contains the command procedure is RBJ21.FASM.CLIST.

The command procedure consists of:

```
proc 1 name
free file(sysin,sysprint)
delete (&name..list,&name..obj)
allocate dataset(&name..asm) file(sysin) old keep
allocate dataset(&name..list) file(sysprint) -
      block(132) space(300,100)
allocate dataset(&name..obj) file(syspunch) block(80) -
      space(100,50)
allocate file(sysut1) space(3,1) cylinders new delete
allocate file(sysut2) space(3,1) cylinders new delete
allocate file(sysut3) space(3,1) cylinders new delete
allocate file(syslib) da('d82ljp1.tso.macro',
      'sys1.maclib') shr
call 'sys1.linklib(ifox00)' 'deck,noobj,rent'
free file(sysut1,sysut2,sysut3,sysin,sysprint, -
      syspunch,syslib)
allocate file(sysin) da(*)
allocate file(sysprint) da(*)
```

Note: If the symbolic value must be immediately followed by a period, the symbolic value must end with a period.

The module to be assembled is "TGETASIS".

You want to have the names of the commands in the command procedure displayed at your terminal as they are executed.

```
exec fasm 'tgetasis' list
```

The display at your terminal will be similar to:

```
EX FASM 'TGETASIS' LIST
FREE FILE(SYSIN,SYSPRINT)
DELETE (TGETASIS.LIST,TGETASIS.OBJ)
IDC0550I ENTRY (A) D82LJP1.TGETASIS.LIST DELETED
IDC0550I ENTRY (A) D82LJP1.TGETASIS.OBJ DELETED
ALLOCATE DATASET(TGETASIS.ASM) FILE(SYSIN) OLD KEEP
ALLOCATE DATASET(TGETASIS.LIST) FILE(SYSPRINT)
    BLOCK(132) SPACE(300,100)
ALLOCATE DATASET(TGETASIS.OBJ) FILE(SYSPUNCH)
    BLOCK(80) SPACE(100,50)
ALLOCATE FILE(SYSUT1) SPACE(3,1) CYLINDERS NEW DELETE
ALLOCATE FILE(SYSUT2) SPACE(3,1) CYLINDERS NEW DELETE
ALLOCATE FILE(SYSUT3) SPACE(3,1) CYLINDERS NEW DELETE
ALLOCATE FILE(SYSLIB) DA('D82LJP1.TSO.MACRO',
    'SYS1.MACLIB') SHR
CALL 'SYS1.LINKLIB(IFOX00)' 'DECK,NOOBJ,RENT'
FREE FILE(SYSUT1,SYSUT2,SYSUT3,SYSIN,SYSPRINT,
    SYSPUNCH,SYSLIB)
ALLOCATE FILE(SYSIN) DA(*)
ALLOCATE FILE(SYSPRINT) DA(*)
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 FASM2.

```
fasm2 tgetasis
```

Example 3

Operation: Enter a fully qualified data set name as a keyword value in an EXEC command value list.

Known:

The procedure named SWITCH is contained in a command procedure library named "MASTER.CLIST" which is allocated as SYSPROC.

The command procedure consists of:

```
PROC 0 DSN1( ) DSN2( )
RENAME &DSN1 TEMPSAVE
RENAME &DSN2 &DSN1
RENAME TEMPSAVE &DSN2
```

If a user whose userid is "USER33" wishes to switch the names of two datasets "MASTER.BACKUP" and "USER33.GOODCOPY", he could invoke the procedure as follows:

Explicit form:

```
exec 'master.clist(switch)' +
     'dsn1(''''''master.backup'''''') +
     dsn2(goodcopy)'
```

Extended implicit form:

```
%switch dsn1('''master.backup''') dsn2(goodcopy)
```

Note that when the implicit forms are used the specification of quoted strings in the value list is made simpler since the value list itself is not a quoted string.

FREE Command

Use the FREE command to release (deallocate) previously allocated data sets that you no longer need. You can also use this command to change the output class of SYSOUT data sets, to delete attribute lists, and to change the data set disposition specified with the ALLOCATE command.

There is a maximum number of data sets that may be allocated to you at any one time. The allowable number must be large enough to accommodate:

- Data sets allocated via the LOGON and ALLOCATE commands
- Data sets allocated dynamically by the system's command processors

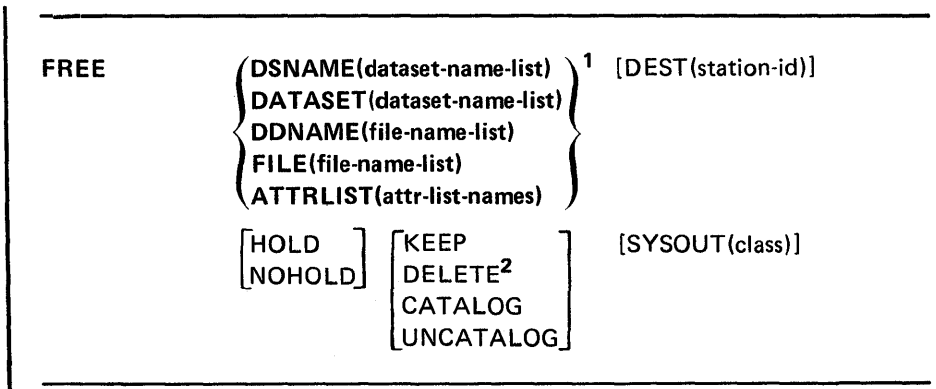
The data sets allocated by the LOGON and ALLOCATE commands are not freed automatically. To avoid the possibility of reaching your limit and being denied necessary resources, you should use the FREE command to release these data sets when they are no longer needed.

When a SYSOUT data set is freed, it is immediately available for output processing, either by the job entry subsystem (not-held data sets) or by the OUTPUT command (held data sets).

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

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.

Note: Data sets that are dynamically allocated by a command processor are not automatically freed when the command processor terminates. You must explicitly free dynamically allocated data sets.



[1] Choose one *or more* of these parameters within braces.

[2] DELETE is the only disposition that is valid for SYSOUT data sets.

DATASET or DSNAME(data-set-name-list)
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. If you omit this operand, you must specify either FILE or DSNAME or the ATTRLIST operand.

FILE or DDNAME(file-name-list)
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 DSNAME or the ATTRLIST operand.

ATTRLIST(attr-list-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 DSNAME or the FILE or DDNAME operand.

DEST(stationid)
specifies a one-to-seven character name of a remote work station to which the SYSOUT data sets are directed when ready for unallocation. If this keyword is omitted on the FREE command for SYSOUT data sets, the data sets will be directed to the work station specified at the time of allocation.

HOLD
specifies that the data set is to be placed on the HOLD queue. HOLD overrides any HOLD/NOHOLD specification made when the data set was originally allocated and it also overrides the default HOLD/NOHOLD specification associated with the particular SYSOUT class specified.

NOHOLD
specifies that the data set is not to be placed on the HOLD queue. NOHOLD overrides any HOLD/NOHOLD specification made when the data set was originally allocated and it also overrides the default HOLD/NOHOLD specification associated with the particular SYSOUT class specified.

KEEP
specifies that the data set is to be retained by the system after it is freed.

DELETE
specifies that the data set is to be deleted by the system after it is freed. DELETE is not valid for data sets allocated SHR or for members of a PDS. Only DELETE is valid for SYSOUT data sets.

CATALOG
specifies that the data set is to be retained by the system in a catalog after it is freed.

UNCATALOG
specifies that the data set is to be removed from the catalog after it is freed. The data set is still retained by the system.

Note: If HOLD, NOHOLD, KEEP, DELETE, CATALOG, and UNCATALOG are not specified, the specification indicated at the time of allocation remains in effect.

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 or DSNAME and FILE or DDNAME operands will be assigned to this class and placed in the output queue for processing by an output writer. In order to free a file to SYSOUT, the file must have previously been allocated to SYSOUT.

Note: A concatenated data set that was allocated in a LOGON procedure or by the ALLOCATE command can be freed only by entering the ddname on the FILE or DDNAME operand.

Example 1

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

Known:
The data set name: TOC903.PROGA.LOAD

```
free dataset(proga.load)
```

Example 2

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

Known:
The data set names: APRIL.PB99CY.ASM, APRIL.FIRSTQTR.DATA, MAY.DESK.MSG

```
free dataset(pb99cy.asm,firstqtr.data,'may.desk
.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: WIND.MARCH.FORT
The filenames (data definition names) of 4 data sets: SYSUT1 SYSUT3 SYSIN SYSPRINT
The new output class: B

```
free dataset(march.fort) file(sysut1,sysut3,sysin,
sysprint) sysout(b)
```

Example 4

Operation: Delete two attribute lists.

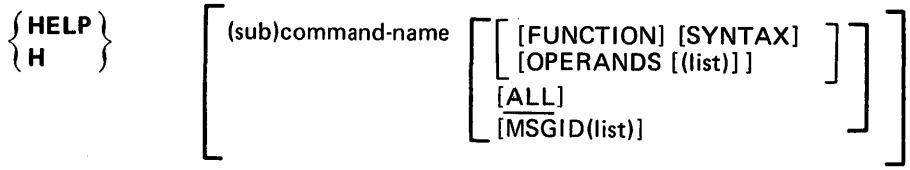
Known:
The names of the lists: DCBPARMS ATTRIBUT

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

HELP Command

Use the HELP command or subcommand to obtain information about the function, syntax, and operands of commands and subcommands and information about certain messages. This reference information is contained within the system and is displayed at your terminal in response to your request for help. By entering the HELP command or subcommand with no operands you can obtain a list of all the TSO commands grouped by function or subcommands of the command you are using.

The HELP command may not be used to get additional information about command procedure statements.



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

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

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

OPERANDS(list-of-operands)
specifies that you want to see explanations of the operands for the command or subcommand. 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 parentheses 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.

MSGID(list)
specifies that you wish to get additional information about VSBASIC messages whose message identifiers are given in the list. Information includes what caused the error and how to prevent a recurrence. The FUNCTION, SYNTAX, OPERANDS or ALL keywords cannot be specified with MSGID.

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

- A brief description of the function of the (sub)command
- The format and syntax for the (sub)command
- A description of each operand

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

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 or subcommand is kept in a member of the partitioned data set. The HELP command or subcommand causes the system to select the appropriate member and display its contents at your terminal.

Figure 10 shows the hierarchy of the sets of information available with the HELP command or subcommand. Figure 10 also shows the form of the command or subcommand 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

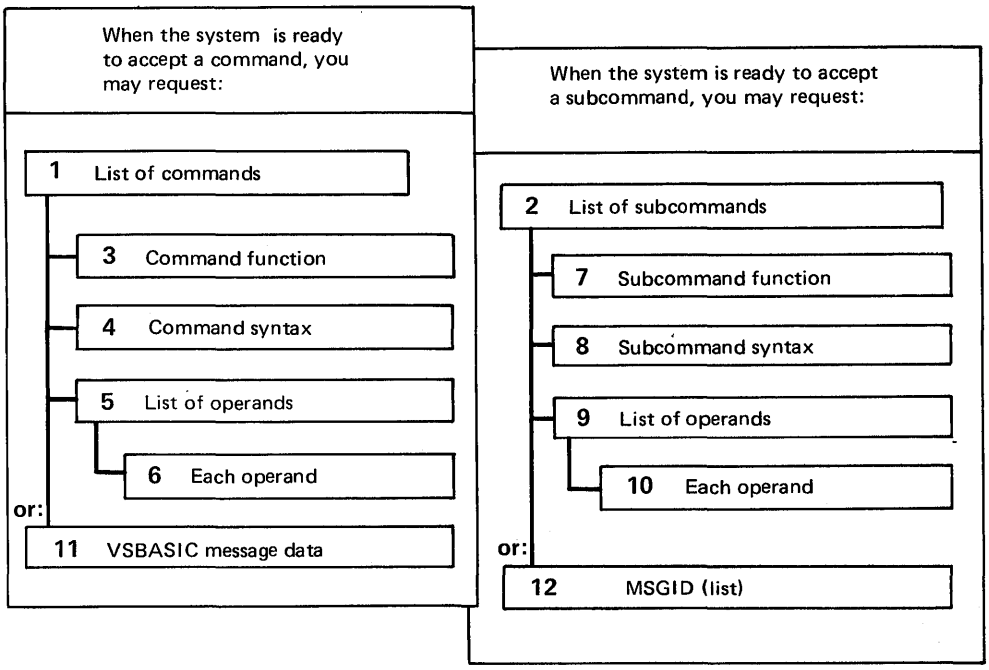
Operation: Have a description of the XREF, MAP, COBLIB, and OVLY operands for the LINK command displayed at your terminal.

```
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
HELP commandname MSGID (list of VSBASIC message ids)	11
EDIT, 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
HELP subcommandname MSGID (list of message ids)	12

Figure 10. Information Available Through the HELP Command

Note: The HELP HELP command is valid only in ready mode.

LINK Command

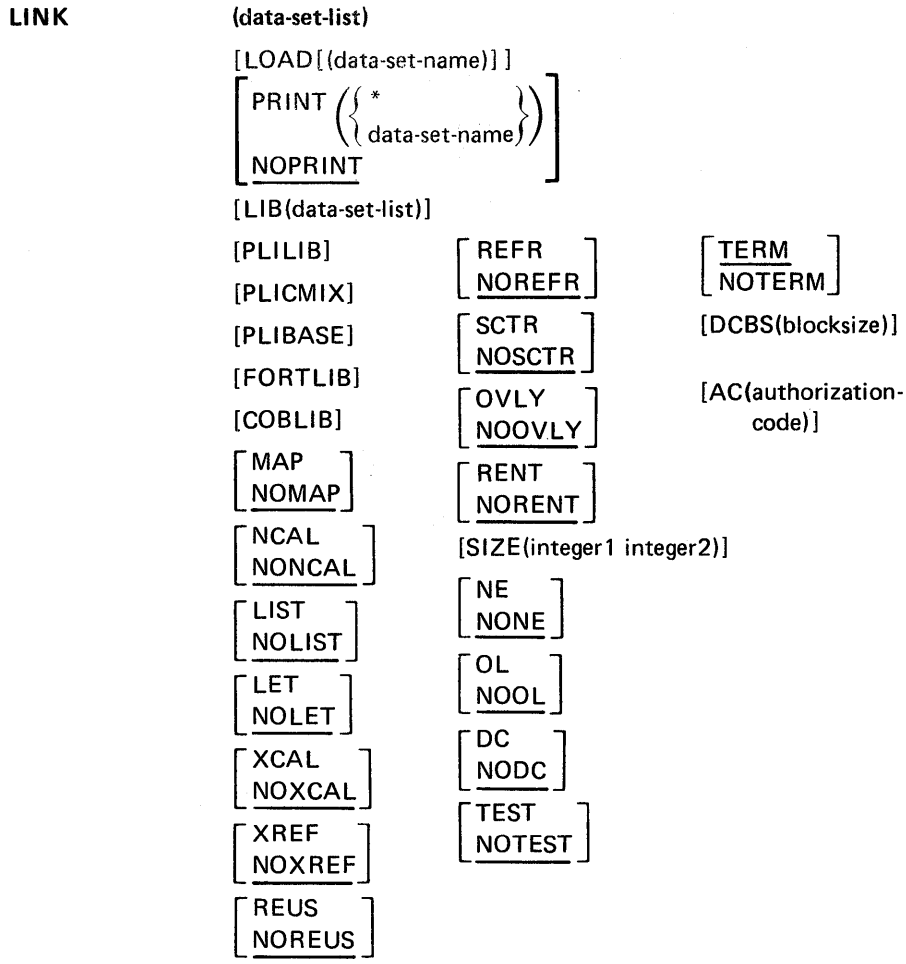
Use the LINK command to invoke the linkage editor service program. Basically, the linkage editor converts one or more object modules (the output modules from compilers) into a load module that is suitable for execution. In doing this, the linkage editor changes all symbolic addresses in the object modules into relative addresses.

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.

Note: You should not link an object module with the TEST option and then attempt to execute the resulting load module in the background because an abnormal termination may result.



(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.

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. This is the default value if you specify the LIST, MAP, or XREF operand. You may substitute an asterisk (*) for the data set name if you want to have the listings displayed at your terminal.

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 object modules referred to by the module being processed; that is, to resolve external references. When you specify more than one name, the names must be separated by a valid delimiter. If you specify more than one name, the data sets are concatenated to the file name of the first data set in the list. For control statements, the first data set in the list must be preallocated with the ddname or file name SYSLIB prior to the LINK command. If you specify more than one name, the data sets are concatenated to the file name of the first data set and lose their individual identity. See *OS/VS2 MVS System Programming Library: Job Management* for details on dynamic concatenation.

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 be be marked reusable. This is 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 real 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 real 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 real 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 real 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 (level E).

NODC
specifies that the load module cannot be reprocessed by the linkage editor (level E). This is the default if both DC and NODC 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. The "blocksize" must be an integer.

AC(authorization-code)
specifies an authorization code (0-255) used to maintain data security.

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: TPB05.GSALESA.OBJ TPB05.GSALESB.OBJ TPB05.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:
TPB05.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: VACID.M33THRUS.OBJ
The name of the existing load module:
VACID.M33PAYLD.LOAD(MOD1)
The name of the standard processor library used for resolving external references in the object module: SYS1.PLILIB
The entry name of the load module is MOD2.
The alias name of the load module is MOD3.
The name of the output load module:
VACID.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 display 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)
    entry mod2
    alias mod3
    (null line)
IKJ76111I END OF CONTROL STATEMENTS
```

LISTALC Command

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. Included in the total 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.

{LISTALC / LISTA} [STATUS] [HISTORY] [MEMBERS] [SYSNAMES]

Notes:

- The LISTALC command without operands will produce a list of all data set names (including those that are not partitioned) which have either been allocated by you or temporarily allocated by previous TSO command processors. This list includes terminal data sets, indicated by the word "TERMINAL" and also attr-list-names created by the ATTRIB command, indicated by the word "NULLFILE".
- LISTA displays a list of data set names allocated by the terminal user. If an asterisk precedes a data set name it indicates that the data set is allocated but marked not-in-use.

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 allocated and the attr-list-names created by the ATTRIB command.
- 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
- An indication as to whether or not the data set has password protection (non-VSAM only)

- The data set organization (DSORG). The listing will contain:

  PS for sequential
  PO for partitioned
  IS for indexed sequential
  DA for direct access
  VSAM for VSAM data entries
  ** for unspecified
  ?? for any other specification

Note: Use the LISTCAT command for further information pertaining to VSAM data entries.

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 the 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

ALLOCATION MUST BE FREED BEFORE RESOURCES CAN BE
RE-USED
 EDTDUMY3
 SYSIN
 SYSPRINT

READY
```

Example 4

Operation: List the names of all your active attribute lists (allocated with ATTRIB command).

```
lista status
```

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

```
lista status
--DDNAME---DISP--
SYS1.LPALIB2
    STEPLIB KEEP
SYS1.UADS
    SYSUADS KEEP
SYS1.BRODCAST
    SYSLBC KEEP
TERMFILE SYSIN
TERMFILE SYSPRINT
*SYS1.HELP
    SYS00005 KEEP,KEEP
D95BAB1.SEPT30.ASM
    SYS00006 KEEP,KEEP
NULLFILE A
NULLFILE B
READY
```

LISTBC Command

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.

{**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 to receive all messages.

```
listbc
```

Example 2

Operation: Specify that you want to receive only the messages intended for all terminal users.

```
listbc nomail
```

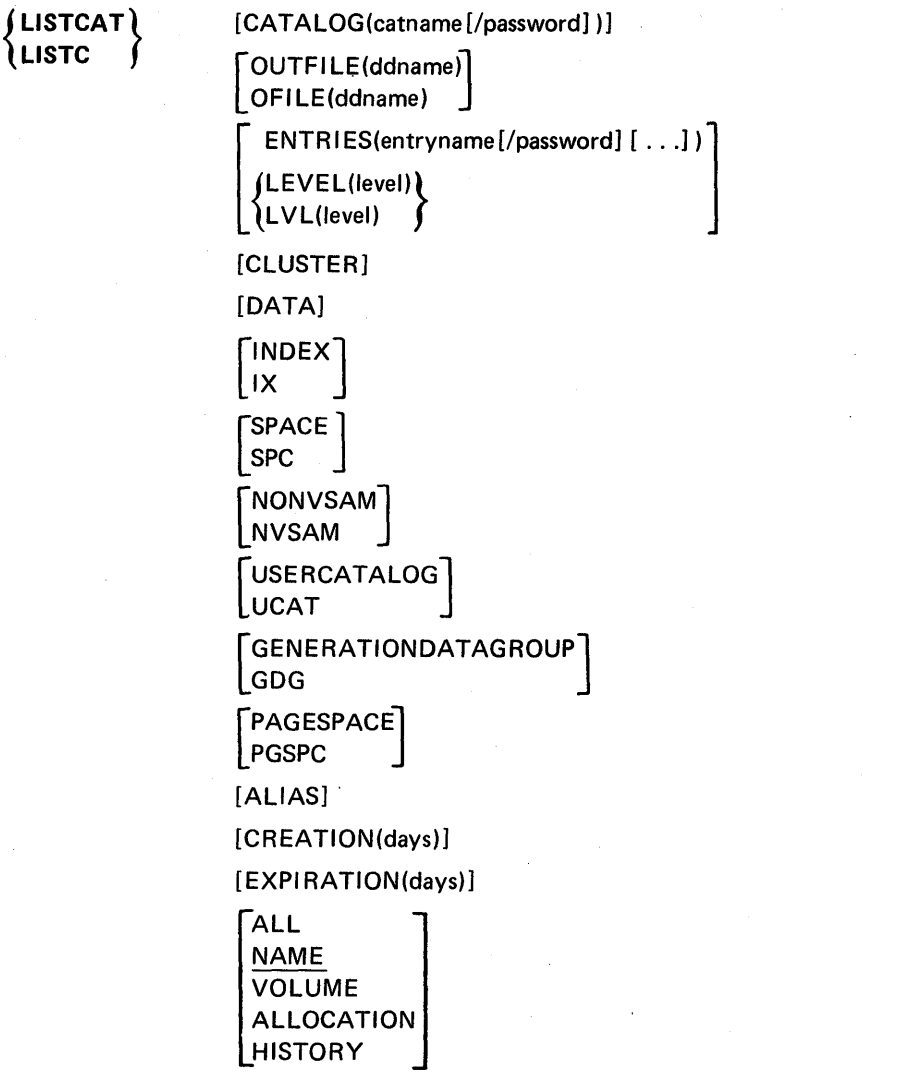
LISTCAT Command

The LISTCAT command is used to list entries from a catalog. The entries listed can be selected by name or entry type, and the fields to be listed for each entry can additionally be selected.

For MVS, the original TSO LISTCAT command has been replaced by an Access Method Services command of the same name. The explanations below provide the information required to use these services for normal TSO operations. The TSO user who wants to manipulate VSAM objects or to use the other Access Method Services from the terminal should refer to *OS/VS2 Access Method Services*. For error message information, see *OS/VS Message Library: VS2 System Messages*.

The LISTCAT command supports unique operand abbreviations in addition to the usual abbreviations produced by truncation. The syntax and operand explanations show these unique cases.

Note: When LISTCAT is invoked and no operands are specified, the userid or the prefix specified by the PROFILE command becomes the highest level of entryname qualification. Only those entries associated with the userid are listed.



CATALOG(catname[/password])
specifies the name of the catalog that contains the entries that are to be listed. When CATALOG is coded, only entries from that catalog are listed.

catname
is the name of the catalog.

password
specifies the read level or higher level password of the catalog that contains entries to be listed. When the entries to be listed contain information about password-protected data sets, a password must be supplied either through this parameter or through the ENTRIES parameter. If passwords are to be listed, you must specify the master password.

OUTFILE(ddname) or OFILE(ddname)
specifies a data set other than the terminal to be used as an output data set. The ddname may correspond to the name specified for the FILE operand of the ALLOCATE command. The data can be listed when the file is freed. The ddname identifies a DD statement that in turn identifies the alternate output data set. If OUTFILE is not specified, the entries are displayed at the terminal.

The normal output data set for listing is SYSPRINT. The default parameters of this data set are:

- Record format: VBA
- Logical record length: 125, that is, 121+4
- Block size: 629, that is, 5 x (121+4)+4

Print lines are 121 bytes in length. The first byte is the ANSI control character. The minimum specifiable LRECL is 121 (U-format records only). If a smaller size is specified, it is overridden to 121.

It is possible to alter the above defaults through specification of the desired values in the DCB parameter of the SYSPRINT statement. The record format, however, cannot be specified as F or FB. If you do specify either one, it is changed to VBA.

In several commands you have the option of specifying an alternte output data set for listing. If you do speicfy an alternate, you must specify DCB parameters in the referenced DD statement. When specifying an alternate output data set, you should not specify F or FB record formats.

ENTRIES(entryname[/password])
specifies the names of the entries to be listed. If neither ENTRIES nor LEVEL is coded, only the entries associated with the user's userid are listed. See *OS/VS2 Access Method Services*.

entry name
specifies the names or generic names of entries to be listed. Entries that contain information about catalogs can be listed only by specifying the name of the master or user catalog as the entry name. The name of a data space can be specified only when SPACE is the only type specified. If a volume serial number is specified, SPACE must be specified.

Note: A qualified name may be made into a generic name by substituting an asterisk (*) for one qualifier. For example, A.* specifies all two-qualifier names that have A as first qualifier; A.*.C specifies all three-qualifier names that have A for first qualifier and C for third qualifier.

password
specifies a password when the entry to be listed is password protected and a password was not specified through the CATALOG parameter. The password must be the read or higher level password. If protection attributes are to be listed, you must supply the master password; if no password is supplied, the operator is prompted for each entry's password. Passwords cannot be specified for non-VSAM data sets, aliases, generation data groups, or data spaces.

LEVEL(level) or LVL(level)
specifies the level of entry names to be listed. If neither LEVEL nor ENTRIES is coded, only the entries associated with the user's userid are listed.

CLUSTER
specifies that cluster entries are to be listed. When the only entry type specified is CLUSTER, entries for data and index components associated with the clusters are not listed.

DATA
specifies that entries for data components, excluding the data component of the catalog, are to be listed. If a cluster's name is specified on the ENTRIES parameter and DATA is coded, only the data-component entry is listed.

INDEX or IX
specifies that entries for index components, excluding the index component of the catalog, are to be listed. When a cluster's name is specified on the ENTRIES parameter and INDEX is coded, only the index-component entry is listed.

SPACE or SPC
specifies that entries for volumes containing data spaces defined in this catalog are to be listed. Candidate volumes are included. If entries are identified by entryname or level, SPACE can be coded only when no other entry-type restriction is coded.

NONVSAM or NVSAM
specifies that entries for non-VSAM data sets are to be listed. When a generation data group's name and NONVSAM are specified, the generation data sets associated with the generation data group are listed.

USERCATALOG or UCAT
specifies that entries for user catalogs are to be listed. USERCATALOG is applicable only when the catalog that contains the entries to be listed is the master catalog.

GENERATIONDATAGROUP or GDG
specifies that entries for generation data groups are to be listed.

PAGESPACE or PGSPC
specifies that entries for page spaces are to be listed.

ALIAS
specifies that entries for alias entries are to be listed.

CREATION(days)
specifies that entries are to be listed only if they were created no later than that number of days ago.

EXPIRATION(days)
specifies that entries are to be listed only if they will expire no later than the number of days from now.

ALL/NAME/VOLUME/ALLOCATION/HISTORY
specifies the fields to be included for each entry listed. If no value is coded, NAME is the default.

ALL
specifies that all fields are to be listed.

NAME
specifies that the names of the entries are to be listed. The default will be NAME.

VOLUME
specifies that the name, owner identification, creation date, expiration date, entry type, volume serial numbers and device types allocated to the entries are to be listed. Volume information is not listed for cluster entries (although it is for the cluster's data and index entries), aliases, or generation data groups.

ALLOCATION
specifies that the information provided by specifying VOLUME and detailed information about the allocation are to be listed. The information about allocation is listed only for data and index component entries.

HISTORY
specifies that the name, owner identification, creation date, and expiration date of the entries are to be listed.

LISTDS Command

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 or a drum.
- The logical record length (LRECL), the blocksize (BLKSIZE) and for non-VSAM data sets, the record format (RECFM) of the data set.
- The data set organization (DSORG); VSAM for VSAM data entries.

  The data set organization is indicated as follows:

  PS for sequential
  PO for partitioned
  IS for indexed sequential
  DA for direct access
  VSAM for VSAM data entries
  ** for unspecified
  ?? for any other specification

  Note: Use the LISTCAT command for further information on a VSAM data entry.
- Directory information for members of partitioned data sets if you specify the data set name in the form *datasetname(membername).*
- Creation date, expiration date, and, for non-VSAM only, security attributes.
- File name and disposition.
- Non-VSAM data set control blocks (DSCB).

Note: Data sets that are dynamically allocated by the LISTDS command processor are not automatically freed when the command processor terminates. You must explicitly free dynamically allocated data sets.

INSERT29*10.6

(data-set-list)
specifies one or more data set names. 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. The names in the data set list may contain a single asterisk in place of any level except the first. When this is done, all cataloged data sets whose names begin with the specified qualifiers are listed. For example, A.*.C specifies all three-qualifier names that have A for first qualifier and C for third qualifier.

Note: Alias data set names are not to be used with this command.

STATUS
specifies that you want the following additional information:

- The 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 and to find out whether or not the non-VSAM 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 non-VSAM data sets on direct access devices. The list will be in hexadecimal notation.

CATALOG
specifies the user catalog that contains the names in the data set list. CATALOG is required only if the names are in a catalog other than STEPCAT or the catalog implied by the first-level qualifier of the name.

LEVEL
specifies that the names in the data set list are to be high-level qualifiers. All cataloged data sets whose names begin with the specified qualifiers are listed. If LEVEL is specified, the names cannot contain asterisks.

Note: Only one data set list may be specified with the LEVEL option.

Example

Operation: List the basic attributes of a particular data set.

Known:
The data set name: ZALD58.CIR.OBJ

```
listds cir
```

The display at your terminal will be similar to the following:

```
listds cir
ZALD58.CIR.OBJ
--RECFM-LRECL-BLKSIZE-DSORG
 FB 80 80 PS
--VOLUMES--
 D95LIB
READY
```

LOADGO Command

Use the LOADGO command to load a compiled or assembled program into real 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. 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.

```
{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 real 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 real 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

```
load 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. When you enter the LOGOFF command, the system frees all the data sets allocated to you; data remaining in storage will be lost.

If you intend to enter the LOGON command immediately to begin a new session using different attributes, you are not required to LOGOFF. Instead, you can just enter the LOGON command as you would enter any other command.

Note: If your terminal is a systems network architecture (SNA) terminal that uses TSO/VTAM, you may be required to use a format different from the one described here. Your system programmer should provide you with this information.

LOGOFF [DISCONNECT / HOLD[1]]

DISCONNECT
specifies that the line is to be disconnected following logoff. This is the default if no operand is specified.

HOLD[1]
specifies that the line is not to be disconnected following logoff.

Example 1

Operation: Terminate your terminal session.

```
logoff
```

---

[1] Not supported with terminals that use TSO/VTAM.

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 be optional at your installation)
- A procedure name (may be optional at 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.

If you are a RACF-defined user, your installation will assign you a RACF password and a GROUP name (optional). As with your userid, this information must be supplied to the system via the LOGON command in order to start and control your time sharing session as a RACF-defined user.

For a RACF user of TSO, the password level in the UADS data set is bypassed and only the password in the RACF data set is compared for user verification during LOGON processing. The remainder of the UADS tree structure (such as account number and procedure name) is enforced. Because the password level in the UADS tree structure is bypassed, a RACF user must supply an account number if the user has more than one account number defined in the UADS data set.

For TSO users who are not defined to RACF, there is no change in the TSO LOGON processing.

Note: If your terminal uses TSO/VTAM, you may be required to use a format different from the one described here. Your system programmer should provide you with this information.

LOGON **user-identity** [/password[/newpassword]]
[ACCT(account)]
[PROC(procedure)]
[SIZE(integer)]
[NOTICES | NONOTICES]
[MAIL | NOMAIL]
[PERFORM(value)]
[RECONNECT]
[GROUP(name)]
[OIDCARD]

---

user-identity/password/newpassword
specifies your user identification and, if required, a valid password or new password. Your 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 are not RACF defined. If you are RACF defined, you must enter the password defined in the RACF data set as the value for password. Newpassword specifies the password that is to replace the current password. Newpassword must be separated from the password by a slash(/) and, optionally, one or more standard delimiters (tab, blank, or comma). The newpassword operand is one to eight alphameric characters in length. This operand is ignored for non-RACF-defined users. (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 maximum region size allowed for a conditional GETMAIN during the terminal session. 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 during LOGON processing.

MAIL
specifies that you want messages intended specifically for you to be displayed at your terminal during LOGON processing. 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 during LOGON processing.

PERFORM(value)
specifies the performance group to be used for the terminal session. The value must be an integer from 1-255. The performance group entered must be defined for you in the user attribute data set (UADS). The default value is performance group 2.

RECONNECT
specifies that you want to re-LOGON after your line has been disconnected. If a password was specified in the disconnected session, the same password must be specified with the RECONNECT option. Any operands other than userid and password will be ignored if RECONNECT is specified.

GROUP(name)
specifies a one-to-eight character ID composed of alphameric and/or national characters, the first of which must be alphabetic or national. This operand is valid only for RACF users. It will be ignored for users not defined to RACF.

OIDCARD
specifies that the operator identification card is to be prompted for after the LOGON command has been entered. This operand is valid only for RACF defined users.
If you are not defined to RACF and enter this keyword, you will be prompted for an operator identification card. However, any data you enter will be ignored. You may also enter a null line in response to the prompt.

Example 1

Operation: Initiate a terminal session.

Known:
Your user identification and password: WRRID/23XA$MBT
Your installation does not require an account number or procedure name for LOGON.

```
logon wrrid/23xa$mbt
```

Example 2

Operation: Initiate a terminal session.

Known:
Your user identification and password: WRRID/MO@
Your account number: 288104
The name of a cataloged procedure: TS951
You do not want to receive any broadcast messages.
Your real storage space requirement: 90K bytes

```
logon wrrid/mo@ acct(288104) proc(ts951)-
    size(90) nonotices nomail
```

Use the PROFILE command or subcommand of EDIT to establish, change, or list 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 (on some terminals).
- Specify whether or not prompting is to occur.
- Specify the frequency of prompting under the EDIT command.
- 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.

Note: The syntax and function of the PROFILE subcommand of EDIT is the same as that of PROFILE.

Initially, a user profile is prepared for you when arrangements are made for you to use the system. The authorized system programmer creates your userid and your user profile. 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 shown in Figure 11 provide for the character-delete and the line-delete control characters depending 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
IBM 3270 Information Display System	None	None
IBM 3767 Communication Terminal	None	None
IBM 3770 Data Communication System	None	None
Teletype* Model 33	**	**
Teletype* Model 35	**	**

* Trademark of Teletype Corporation.
** Refer to *TSO Terminal User's Guide.*

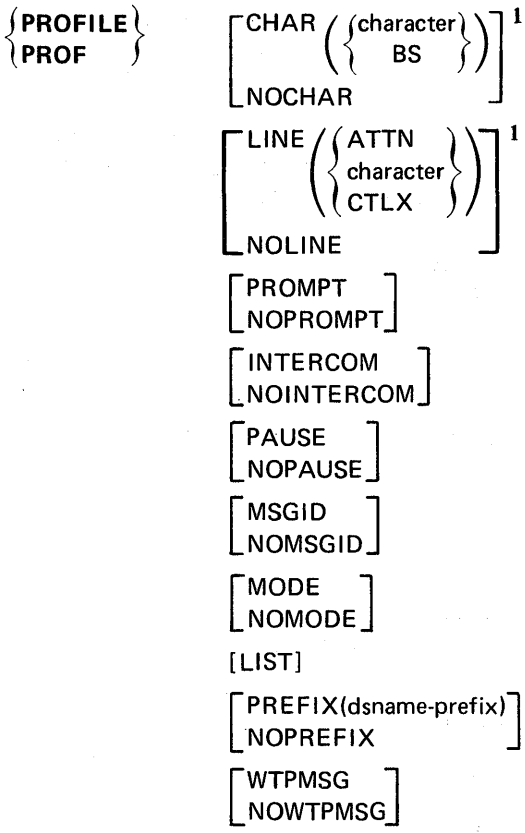
Figure 11. System Defaults for Control Characters

Note: If deletion characters, prompting, and message activity are not what you expect, check your profile by displaying it with LIST operand.

Change your 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. If PROFILE

changes do not remain from session to session, your installation may have a LOGON pre-prompt exit that is preventing the saving of any changes in the UPT. Verify this with your system programmer.

If no operands are entered on the PROFILE command, the current user profile will be displayed.



CHAR(character)[1]
specifies the EBCDIC 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 parentheses because these characters are used to enter commands. You should not specify terminal-dependent characters which do not translate to a valid EBCDIC character.

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 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.

[1] Not supported with terminals that use TSO/VTAM.

CHAR(BS)[1]
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[1]
specifies that no control character is to be used for character deletion.

LINE(character)[1]
specifies a control character that you want to use to tell the system to delete the current line.

LINE(ATTN)[1]
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.

Note: If an invalid character and/or line delete control character is entered on the PROFILE command, an error message will inform the user which specific control character is invalid; the character or line delete field in the user profile table will not be changed. You may continue to use the old character or line delete control characters.

LINE(CTLX)[1]
specifies that the X and CCTRL keys (depressed together) on a Teletype[2] 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[1]
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) or an in-storage command list (created via the STACK macro) 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 press the ENTER key.

---

[1] Not supported with terminals that use TSO/VTAM.

[2] Trademark of the Teletype Corporation.

NOPAUSE
specifies that you do not want to be prompted for a question mark or ENTER. This is the default value 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.

LIST
specifies that the characteristics of the terminal user's profile be listed at the terminal. If other operands are entered with LIST, the characteristics of the user's profile will be changed first, and then the new profile will be listed.

Notes:

1. After a new userid is created and before the character-delete and/or line-delete control character is changed, entering PROFILE LIST will result in CHAR(0) and LINE(0) being listed. This indicates that the terminal defaults for character-delete and line-delete control characters will be used.
2. Although you may receive RECOVER/NORECOVER as an option for this operand, these options can only be used with the command package program product (5740-XT6). However, if the command package is installed, you must be authorized to use the RECOVER option.

MODE
specifies that a mode message is requested at the completion of each subcommand of EDIT.

NOMODE
specifies that, when this mode is in effect, the mode message (E or EDIT) will be issued after a SAVE, RENUM or RUN subcommand is issued and also when changing from input to edit mode.

PREFIX(dsname-prefix)
specifies a prefix which will be appended to all non-fully qualified dsnames. The prefix is composed of one-to-seven alphameric characters that begin with an alphabetic or national character.

NOPREFIX
specifies no prefixing of dsnames by any qualifier will be performed.

Note: The default prefix in the foreground is the userid. No prefixing of data set names is the default in the background.

WTPMSG
specifies that the user wishes to receive all write-to-programmer messages at his terminal.

NOWTPMSG
specifies that the user does not want to receive write-to-programmer messages. 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 ENTER key, you will actually enter:

```
abc.
```

PROTECT Command

Use the PROTECT command to prevent unauthorized access to your non-VSAM data set. (Use the Access Method Services ALTER and DEFINE commands to protect your VSAM data set. These commands are described in *OS/VS2 Access Method Services*.) This command establishes or changes:

- The passwords that must be specified to gain access to your data
- 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:

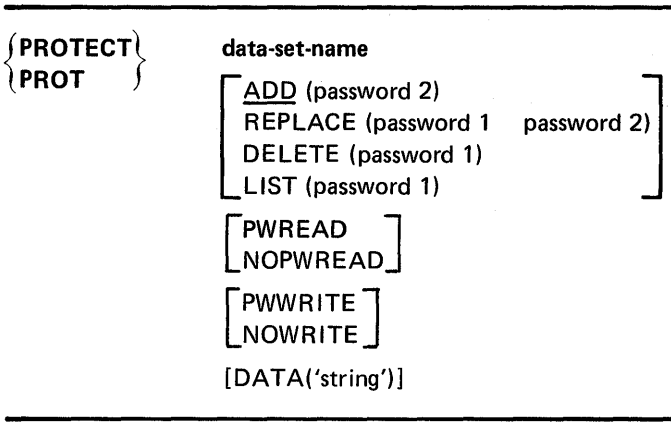
OPERAND	DEFAULT VALUE	
PWREAD	PWREAD	PWWRITE
NOPWREAD	NOPWREAD	PWWRITE
PWWRITE	NOPWREAD	PWWRITE
NOWRITE	PWREAD	NOWRITE

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.

Note: If the data set is not cataloged, the user must specify the fully qualified name. For example:

```
protect 'userid.dsn.qual' list(password)
```

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 increased 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; 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: ROBID.SALES.DATA
The password: L82GRIFN
The type of access allowed: PWREAD PWWRITE
The logon id was: ROBID

```
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: ROBID.NETSALES.DATA
The existing password: MTG@AOP
The new password: PAO$TMG
The control password: ELHAVJ
The logon id was: ROBID

```
prot netsales.data/elhavj replace(mtg@aop,pao$tmg)
```

Example 3

Operation: Delete one of several passwords.

Known:
The name of the data set: ROBID.NETGROSS.ASM
The password: LETGO
The control password: APPLE
The logon id was: ROBID

```
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: ROBID.BILLS.CNTRLA
The password required: D#JPJAM

```
protect 'robid.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: ROBID.PROJCTN.LOAD
The new type of access: NOPWREAD PWWRITE
The existing password: DDAY6/6
The control password: EEYORE
The logon id was: ROBID

```
protect projctn.load/eeyore replace(dday6/6)-
nopwread pwwrite
```

RENAME Command

Use the RENAME command to:

- Change the name of a non-VSAM 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.

Notes:

1. The Access Method Services ALTER command changes the name of VSAM data sets and is described in *OS/VS2 Access Method Services.*.
2. When a password protected data set is renamed, the data set does not retain the password. You must use the PROTECT command to assign a password to the data set before you can access it.

{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.

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.

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.

The RENAME command should not be used to create an alias for a linkage-editor created load module.

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 non-VSAM 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)".

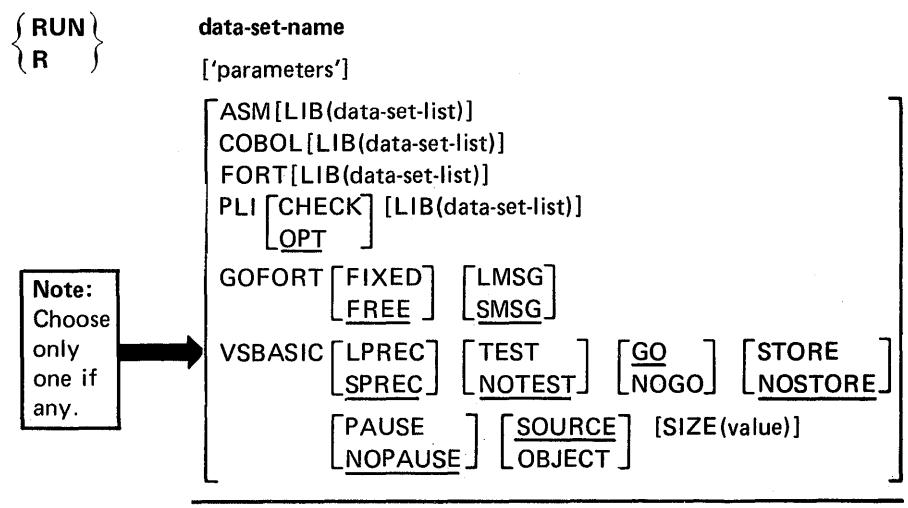
```
REN 'ELIZBETH(LIZ)' (SUZIE) ALIAS
```

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. Figure 12 shows which program product is selected to process each type of source statement.

If your program or data set contains statements of this type (see EDIT):	Then the following program product (or equivalent) can be used:
ASM	TSO ASM Prompter
COBOL	TSO COBOL Prompter and OS Full American National Standard COBOL Version 3 or Version 4 Compiler
FORTGI	TSO FORTRAN Prompter and FORTRAN IV (GI) Compiler
GOFORT	Code and Go FORTRAN
PLI	PL/I Checkout Compiler or PL/I Optimizing Compiler
VSBASIC	TSO VSBASIC Prompter
You can use the CONVERT command to convert Code and Go FORTRAN free-form statements to a form suitable for the FORTRAN compiler.	

Figure 12. Source Statement/Program Product Relationship

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.

LIB(data-set-list)
specifies the library or libraries that contain subroutines needed by the program you are running. These libraries are concatenated to the default system libraries and passed to the loader for resolution of external references. This operand is valid only for the following data set types: ASM, COBOL, FORT, and PLI (Optimizer).

COBOL
specifies that the TSO COBOL Prompter and the OS Full American National Standard COBOL (Version 3 or Version 4) 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 (G1) 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.

PLI
specifies that the PL/I Prompter and either the PL/I Optimizer compiler or the PL/I Checkout compiler are 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. If you omit this operand, the OPT operand is the default value.

OPT
specifies the PL/I Optimizing compiler. This is the default value if both CHECK and OPT are omitted.

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 VSBASIC program product.

NOTEST
specifies that the TEST function is not desired. This is the default value, and is valid only for the VSBASIC program product.

LMSG
specifies that the long form of the diagnostic messages are to be provided. This operand is applicable to the Code and Go FORTRAN program product. The default value is SMSG.

SMSG
specifies that the short form of the diagnostic messages is to be provided. This operand is applicable to the Code and Go FORTRAN program product.

LPREC
specifies that long precision arithmetic calculations are required by the program. This operand is valid only for the VSBASIC program product.

SPREC
specifies that short precision arithmetic calculations are adequate for the program. This operand is valid only for the VSBASIC program product and is the default value.

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.

VSBASIC
specifies that the VSBASIC program product is to be invoked to process the source program.

GO
specifies that the program is to receive control after compilation. This is the default if neither GO nor NOGO are specified. This operand is valid only for VSBASIC.

NOGO
specifies that the program will not receive control after compilation. This operand is valid only for VSBASIC.

STORE
specifies that the compiler is to store an object program. This operand is valid only for VSBASIC.

NOSTORE
specifies that the compiler is not to store an object program. This is the default if neither STORE nor NOSTORE are specified. This operand is valid only for VSBASIC.

PAUSE
specifies that the compiler is to prompt to the terminal between program chains. This operand is valid only for VSBASIC.

NOPAUSE
specifies no prompting between program chains. This is the default if neither PAUSE nor NOPAUSE is specified. This operand is valid only for VSBASIC.

SOURCE
specifies that new source code is to be compiled. This is the default if neither SOURCE nor OBJECT is specified. This operand is valid only for VSBASIC.

OBJECT
specifies that the data set name entered is a fully-qualified name of an object data set to be executed by the VSBASIC compiler.

SIZE(value)
specifies the number of thousand-byte blocks of user area where value is an integer of one-to-three digits. This operand is valid only for VSBASIC.

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, PLI, GOFORT, or VSBASIC) 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.

The RUN command uses standard library names, such as SYS1.FORTLIB and SYS1.COBLIB, as the automatic call library. This is the library searched by the linkage editor to locate load modules referred to by the module being processed for resolution of external references.

Note: RUN causes other commands to be executed from an in-storage list. If an error occurs, one of these commands may issue a message that has additional levels of information. This additional information will not be available to the user unless the PAUSE option is indicated in the user's profile. The PAUSE option is described in the section titled, "PROFILE command".

Example 1

Operation: Compile, load, and execute a source program composed of VSBASIC statements.

Known:
The name of the data set containing the source program is DDG39T.MNHRS.VSBASIC.

```
run mnhrs.vsbasic
```

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.

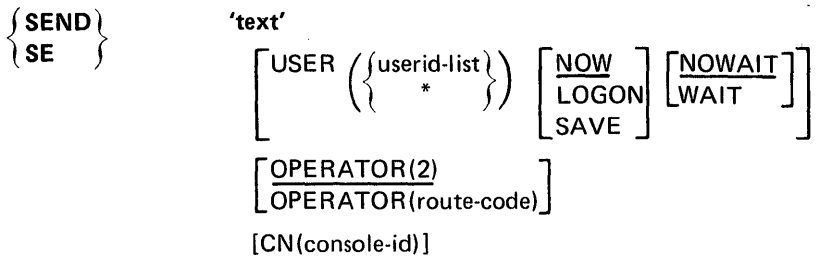
Known:
The data set name TRAJECT.MISSILE FORTRAN statements conform to the standard format. Complete diagnostic messages are needed.
Parameters to be passed to the program are: 50 144 5000

```
run 'traject.missile' '50 144 5000' gofort fixed lmsg
```

SEND Command

Use the SEND command or SEND subcommand of EDIT 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.

Note: The syntax and function of the SEND subcommand of EDIT is the same as that of SEND 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(userid-list)
specifies the user identification of one or more terminal users who are to receive the message. A maximum of 20 identifications can be used.

USER(*)
specifies that the message will be sent to the userid associated with the issuer of the SEND command. If an '*' is used with a SEND command in a command procedure, the message will be sent to the user executing the command procedure. If used with the SEND command at a terminal, an '*' will cause the message to be sent to the same terminal.

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 NOW, LOGON, and SAVE 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.

SAVE
specifies that the message text is to be entered in the mail section of SYS1.BRODCAST without being sent to any user. Messages stored in the broadcast data set can be retrieved by using either LISTBC or LOGON commands.

WAIT
specifies that you will wait until system output buffers are available for all specified logged-on terminals. This ensures that the message will be received by all specified logged-on users, but it also means that you may be locked out until all such users have received the message.

NOWAIT
specifies that you do not want to wait if system output buffers are not immediately available for all specified logged-on terminals. You will be notified of all specified users who did not receive the message. If you specified LOGON, mail will be created in the SYS1.BRODCAST data set for the specified users whose terminals are busy or who have not logged-on. NOWAIT is the default value if neither WAIT nor NOWAIT is specified.

OPERATOR(route-code)
specifies that you want the message sent to the operator indicated by the route-code. If you omit the route-code, 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.

CN(console-id)
specifies that the message is to be queued to the indicated operator console. The value for "console-id" must be an integer between 0-64.

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: If you have data set 'mylib.load' allocated, please free it. I need it to run my program.
The user identification for the terminal users: JANET5
LYNN6
The message is important and you want to make sure the specified user gets it now.

```
send 'if you have data set "mylib.load" allocated, -
please free it. i need it to run my program.' -
user(janet5,lynn6) wait
```

Example 3

Operation: Send a message that is to be delivered to 'BETTY7' when she begins her terminal session or now if she is currently logged on.

Known:

The recipients's user identification: BETTY7

The message: Is your version of the simulator ready?

If her terminal is busy, you want to put the message into the SYS1.BRODCAST data set. There is no rush for her to get it and respond.

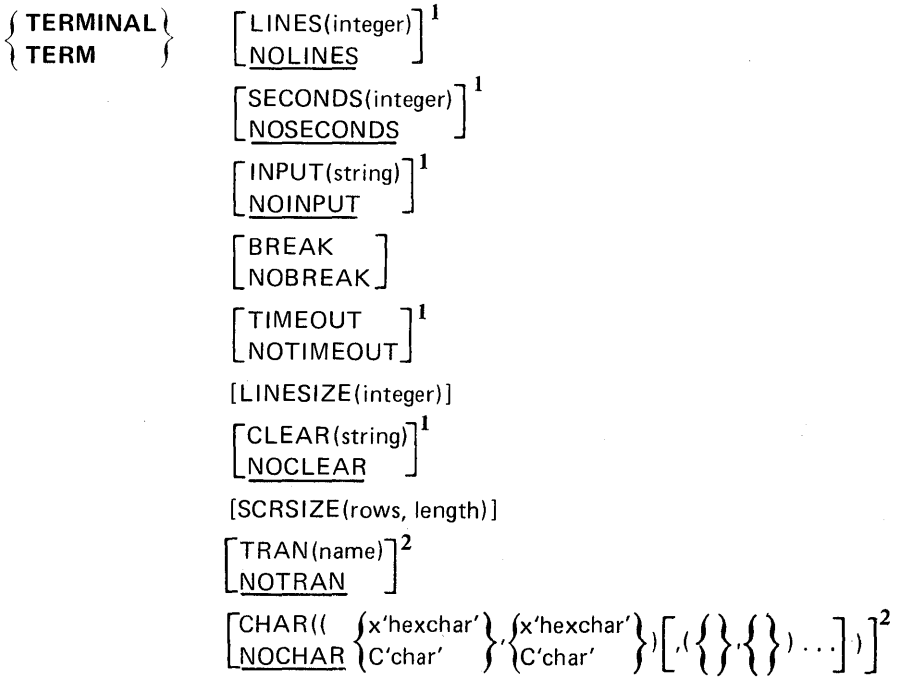
```
send 'is your version of the simulator ready?' -
user(betty7) logon nowait
```

TERMINAL Command

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 command is not allowed as a TSO command in the background.

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.

If your session is interrupted by a line disconnection and you relogon via the LOGON RECONNECT, you must redefine all previously defined terminal characteristics. The reason for the redefinition is that all records for defined data are lost as a result of the line disconnection.



LINES(integer)[1]
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.

[1] Not supported with terminals that use TSO/VTAM.

[2] Not supported with terminals that use TSO/TCAM.

NOLINES[1]
specifies that output line count is not to be used for controlling an attention interruption. This is the default condition.

SECONDS(integer)[1]
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[1]
specifies that elapsed time is not to be used for controlling an attention interruption. This is the default condition.

INPUT(string)[1]
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[1]
specifies that no character string will cause an attention interruption. This is the default condition.

BREAK
specifies, for IBM 3767 and IBM 3770 terminals, that the system can interrupt your input. For other terminals, it 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.

Note: If a command processor for a display device is operating in full-screen mode, TSO/VTAM treats the device as if it were operating in NOBREAK mode. For a more detailed description see, *OS/VS2 TSO Guide to Writing a Terminal Monitor Program or a Command Processor.*

NOBREAK
specifies, for IBM 3767 and IBM 3770 terminals, that the system is not allowed to interrupt you (break in) with a message when you are entering data. For other terminals, it 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[1]
specifies that your terminal's keyboard will lock up automatically after approximately nine to 18 seconds of no input. (This is applicable only to the IBM 1052 Printer-Keyboard without the text timeout suppression feature.)

---

[1] Not supported with terminals that use TSO/VTAM.

[2] Not supported with terminals that use TSO/TCAM.

NOTIMEOUT[1]
specifies that your terminal's keyboard will not lockup automatically after approximately nine to 18 seconds of no input. (This is 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. (This is not applicable to the IBM 2260, 2265, and 3270 Display Stations.) Default values are as follows:

IBM 2741 Communication Terminal	- 120 characters
IBM 1052 Printer-Keyboard	- 120 characters
Teletype 33/35	- 72 characters
IBM 3767 Communication Terminal	- 132 characters
IBM 3770 Communication System	- 132 characters

The integer must not exceed 255.

Note: If LINESIZE (80) is coded with a RECFM equal to U, then the line will be truncated. The byte truncated (the last byte) is reserved for an attribute character.

CLEAR(string)[1]
specifies a character string that, if entered as input, will cause the screen of an IBM 2260, 2265, or 3270 Display Station to be erased. The 'string' must be the only input entered and cannot exceed four characters in length.

NOCLEAR[1]
specifies that you do not want to use a sequence of characters to erase the screen of an IBM 2260, 2265, or 3270 Display Station. This is the default condition.

SCRSIZE(rows,length)
specifies the screen dimensions of an IBM 2260, 2265, or 3270 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. Standard screen sizes are:

rows,length
6,40
12,40
12,80
15,64
24,80
32,80
43,80

Note: The default values for the screen sizes are determined when your installation defines the terminal features.

---

[1] Not supported with terminals that use TSO/VTAM

[2] Not supported with terminals that use TSO/TCAM

TRAN(name)[2]
specifies a load module that contains tables used to translate specific characters you type at the terminal into different characters when they are seen by TSO. Conversely, when these characters are sent by TSO to the terminal, they are retranslated. (Translation of numbers and uppercase letters is not allowed.)
Character translation is especially useful when you are using a correspondence keyboard and would like to type the characters "<", ">", and "|", which are not available on a correspondence keyboard. Translation tables make it possible for you to specify that when you type the characters "[", "]", and "!", TSO interprets them as "<", ">", and "|".

NOTRAN[2]
specifies that no character translation is to take place.

CHAR[2]
specifies one or more pairs of characters, in either hexadecimal or character notation, that replace characters in the translation tables specified by TRAN(name) or in the default translation tables. When the default translate is used, all unprintable characters are set to blanks. The first character of the pair is the character typed, printed, or displayed at the terminal. The second character is the character seen by TSO. (Translation of numbers and uppercase letters is not allowed.) Do not select characters that may be device control characters.

NOCHAR[2]
specifies that all character translations previously specified by CHAR are no longer in effect.

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)
```

---

[2] Not supported with terminals that use TSO/TCAM

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(%)
```

Example 4

Operation: Specify character translation for certain characters not available on an IBM 3767 Communication Terminal with an EBCDIC keyboard.

Known:
Your terminal supports the character translation facility, and you are using the default translation table or a previously specified translation table (that you specified with the TRAN operand). You now want "[" to stand for "<", "]" to stand for ">", and "!" to stand for "|".

```
terminal char((C'[',X'4C'),(C']',X'6E'),(C'!',X'FA'))
```

TEST Command

Use the TEST command to test a program or a command processor for proper execution and to locate programming errors. To use the TEST command and subcommands, you should be familiar with the assembler language and addressing conventions. For best results, the program to be tested should be written in basic assembler language. To use the symbolic names feature of TEST your program should have been assembled and link-edited with the TEST operands.

Note: If the problem program attempts to LOAD, LINK, XCTL, or ATTACH another module from a data set that is not in the list of data sets in link library list concatenation (LNKLST), LPA, or in your JOBLIB, STEPLIB, or TASKLIB libraries, the module will not be found. (To avoid this situation, you may bring the module into virtual storage via the LOAD subcommand of TEST.)

When to Use TEST

There are two basic situations in which you might use the TEST command:

- To test a currently executing program
- To test a program not currently being executed

You might want to test an executing program because it either: terminated abnormally or because you want to check the current environment to see that the program is executing properly.

Note: TEST will be rejected if the terminating or interrupted program is either APF authorized, executing in supervisor state or in a PSW protection key less than 8.

If a program terminates abnormally when not under TEST, you receive a diagnostic message from the terminal monitor program (TMP) followed by a READY message. If you respond to the diagnostic message with anything other than TEST, a question mark (?), or TIME, the TMP terminates your program. However, if you issue the TEST command (and supply no program name), the currently active program remains in storage when the TEST command processor gets control and you can use the TEST subcommands to debug the defective program.

Note: Both the ? and the TIME command can be entered before you issue the TEST command to debug an abnormally terminating program. However, if you want a dump, instead of issuing the TEST command, enter a null line. If either a SYSABEND, SYSMDUMP, or SYSUDUMP file has already been allocated, the null line will result in a dump being printed.

If you want to examine the current environment of an executing program that is not terminating abnormally, enter a single attention interruption. The currently active program remains attached and the TMP responds to your interruption by issuing a READY message. When you issue the TEST command (without a program name) the currently active program remains in storage under the control of the TEST command processor. You can then use the TEST subcommands to examine the current environment.

Note: In the case of either the abend or the attention interruption, you should *not* enter a program name following the TEST command. If you do, you will lose the current in-storage copy of the program, as TEST loads a copy of the specified program.

To test a program not currently executing, enter the TEST command supplying the data set name containing the program to be executed and any other applicable operands. When you use the TEST command to load and execute a program, that program *must* be an object module or a load module suitable for execution.

Prior to and during execution, such as when execution is interrupted at a breakpoint, you can:

- Supply initial values (test data) that you want to pass to the program
- Establish breakpoints at instructions where execution is to 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.)
- Display the contents of registers and virtual storage
- Modify the contents of registers and virtual 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

Note: When running in (supervisor state or in a PSW protection key less than 8, breakpoints will not be honored in any section of your program.

Addressing Conventions Associated with TEST

An address used as an operand for a subcommand of TEST must be one of the following types:

- **Absolute address** - a 1 to 6 hexadecimal digit number followed by a period. An absolute address specifies an absolute virtual storage address.
- **Relative address** - a 1 to 6 hexadecimal digit number preceded by a plus sign (+). This type of address specifies an offset from the currently qualified virtual storage address. See the section titled "Qualified Addresses" for a detailed description of qualified addressing.
- **Symbolic address** - 1 to 8 alphameric characters, the first of which is an alphabetic character. A symbolic address corresponds to a symbol in a program or a symbol defined via the EQUATE subcommand. See the section titled "Qualified Addresses" for a detailed description of qualified symbolic addressing. See the section titled, "Restrictions on the Use of Symbols" for a detailed description on the use of symbols.
- **[module-name].entry-name** - a name within a module capable of being externally referenced, preceded by a period (.) and optionally preceded by a name by which the module is known. An entry name is the symbolic address of an entry point into the module, for example, a

CSECT name. A module name may be the name or alias of a load module or the name of an object module. If the user did not specify a name for an object module when it was loaded by the OS loader, the name TEMPNAME is assigned.

- **Qualified addresses** - You may qualify symbolic or relative addresses to indicate they apply to a particular module and CSECT. To do this you must precede the address by the name of the load or object module and the name of the CSECT. The qualified address must be in the form:

```
modulename.csect.address
```

If the address is to apply to the current module, you need only specify the CSECT name in the following form:

```
csect.address
```

If the address is to apply to the current CSECT within the current module, only the address is necessary; you do not need to qualify the address. The current module and CSECT is initially set to the program being tested. This setting is automatically changed each time a module under a different request block is invoked. This is referred to as *automatic qualification.* (This happens when a module is invoked via ATTACH, XCTL, SYNCH, or LINK. It does *not* happen when a module is loaded, called, or branched to.) The module and/or CSECT used in determining a base location for resolving symbolic and relative addresses may also be changed by using the QUALIFY subcommand.

For example, if the name of the module is OUTPUT, the CSECT is TAXES, and the symbolic address is YEAR77, you would specify either:

```
output.taxes.year77
```

or

```
.taxes.year77
```

if the current module is OUTPUT. You would specify:

```
year77
```

if the current module is OUTPUT and the current CSECT is TAXES. If the module name and CSECT name are the same as above and the address to be qualified is the relative address +4A, you would specify:

```
output.taxes.+4A
```

- **General registers:** You can refer to a general register using the COPY, LIST or assignment-of-value 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 0 through 15. Other references to the general registers imply indirect addressing.

Note: If your program issues the STIMER macro or involves asynchronous interruptions, the contents of your registers may be changed by interruptions even though you are in TEST *subcommand* mode and your program does not get control.

- **Floating-point registers:** You can refer to a floating-point register using the LIST or assignment-of-value subcommand by specifying a decimal integer followed by an E or D. The decimal integer indicates the number of the register and *must be* a zero, two, four, or six. An E indicates a floating-point register with single precision. A D indicates a floating-point register with double precision. The contents of the floating-point register must be assigned using the notation described in section titled "Assignment of Values Function of TEST". You *must not* use floating-point registers for indirect addressing or in expressions.
- **Indirect address** - a address expression, a general register, or the address of a location that contains another address. An indirect address *must* be followed by a percent sign (%). (The percent sign indicates that the address is indirect.) To use a general register as an indirect address, specify a decimal integer (0 through 15) followed by an R and a percent sign. For instance, if you want to refer to data whose address is located in register 7, then you would specify:

  ```
  7r%
  ```

- **Address expression** - an address followed by any number of expression values. Address can be:
  - An absolute address
  - A relative address (unqualified, partially or fully qualified)
  - A symbolic address (unqualified, partially or fully qualified)
  - An indirect address

  An expression value consists of a plus or minus displacement value (a 1-to-6 digit number that may be expressed in either decimal or hexadecimal form) from an address in virtual storage. Following are two examples of address expressions:

  Decimal Example:

  address+14n specifies the location that is 14 bytes past that designated by "address"

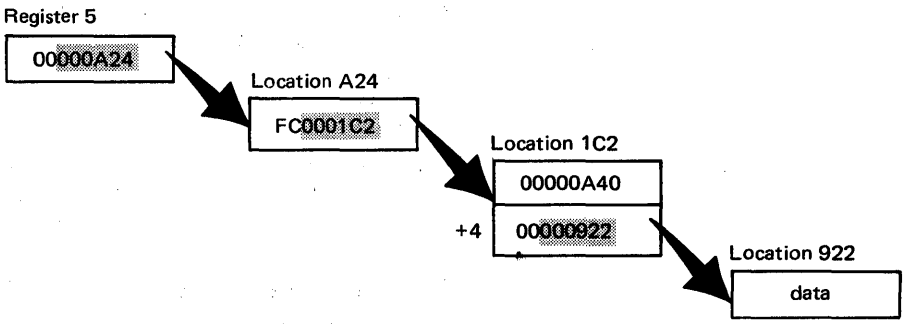
  Hexadecimal Example:

  address+14 specifies the location that is 20 decimal bytes past that designated by "address"

  Note: Decimal displacement (either plus or minus) is indicated by the n following the numeric offset. You can indicate up to 256 levels of indirect addressing by following the initial indirect address with a corresponding number of percent signs. An address expression is specified like this:

  ```
  address {-} value[%...][ {-} value[%...]]...
          {+}              {+}
  ```

For example, the expression 5R%%+4% refers to the data in location 922 below.



Restrictions on Use of Symbols

The TEST command processor can resolve external and internal symbolic addresses only if these addresses are available to it. Within certain limitations, symbolic addresses are available for both object modules (processed by the OS/VS loader) and load modules (fetched by contents supervision).

External Symbols

The TSO TEST user can access external symbols, such as CSECT names, for a program modules if the program was brought into main storage by the TEST command or one of its subtasks. This is the case for the program being tested, any program brought into storage through the tested program, and any program loaded via the LOAD subcommand.

External symbols for CSECT names that are in object modules are available only if the OS/VS loader had enough main storage to build composite external symbol table dictionary (CESD) entries.

Internal Symbols

Internal symbols for load modules can be resolved if the CSECT containing the symbol was assembled with the TEST parameter, the module was link edited with the TEST parameter, and the program was brought into storage by the TEST command or one of its subtasks as previously explained. Names on EQU, ORG, LTORG, CNOP, and DSECT statements *cannot* be resolved.

The TSO TEST user can not access internal symbols for object modules.

Addressing Considerations

If the necessary conditions for symbol processing are not met, you can use absolute, relative, or indirect addressing or you can define symbols with the EQUATE subcommand of TEST.

Symbols within *DSECTs* are available only if the *DSECT* name has been defined with the EQUATE subcommand.

For example, if NAME is a symbol in a DSECT named DATATBL, then to access the data associated with NAME, the user would have to first determine the address to be used as a base address for the DSECT. (This is the address in the register on the assembler USING instruction.) If the address is in register 7 the user may enter:

```
equate datatbl 7r%
```

This will establish addressability to the DSECT, allowing the symbol NAME and all other symbols in the DSECT to be accessed using the symbol.

Note: No symbols are available for a module loaded from a data set, other than SYS1.LINKLIB, which is in LNKLST concatenation.

Examples of Valid Addresses in TEST Subcommands

Below is a list of valid addresses which can be used with subcommands:

Address:	Type of Address:
A23C40.	Absolute
+E4	Relative
5R%	Indirect
NAMES	Symbol within program
.SALES.+26	Partially-qualified relative
14R%+28	Expression
PROFIT.SALES	Module and entry name
+16+10n	Expression
.SALES.NAMES	Partially-qualified symbol
PROFIT.SALES.NAMES+8n	Expression
DATA+10	Expression
.SALES	Entry name
PROFIT.SALES.NAMES	Fully-qualified symbol
6R%+4%+12n%%	Expression
PROFITS.SALES.+CO.	Fully qualified relative

Note: In the above addresses PROFIT is the module name, SALES is the CSECT name and NAMES is the symbol.

```
TEST        ['data-set-name']
            ['parameters']
            [LOAD  ]
            [OBJECT]
            [CP    ]
            [NOCP  ]
```

'data-set-name'
specifies the name of the data set containing the program to be tested. The program must be a load module that is a member of a partition data set or it must be an object module.

A data set name must be specified to test a program that is not currently active. (A currently active program is one that has abnormally terminated or has been terminated by an attention interruption.)

Note: When specifying the data-set-name for TEST, the name should be enclosed by single quotes or the LOAD or OBJECT qualifier will be added to the name specified. If no name is specified, TEMPNAME is the member searched for via the TEST request.

Caution: The program to be tested should not have the name TEST or the name of any existing TSO service routine. For a listing of the existing module names see, *OS/VS2 TSO Terminal Monitor Program and Service Routines Logic*.

'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 virtual storage and in registers.

AT
establishes breakpoints at specified locations.

CALL
initializes registers and initiates processing of the program at a specified address, using the standard subroutine linkage.

COPY
moves data.

DELETE
deletes a load module from virtual storage.

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 virtual storage.

GETMAIN
acquires a specified number of bytes of virtual 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 a virtual storage area or registers.

LISTDCB
lists the contents of a data control block (DCB) (you must specify the address of the DCB).

LISTDEB
lists the contents of a data extent block (DEB) (you must specify the address of the DEB).

LISTMAP
displays a map of the user's virtual storage.

LISTPSW
displays a program status word (PSW).

LISTTCB
lists the contents of the current control block (TCB) (you may specify the address of another TCB).

LOAD
loads a program into virtual storage.

OFF
removes breakpoints.

QUALIFY
establishes the starting or base location for resolving symbolic or relative addresses; resolves identical external symbols within a load module.

RUN
terminates TEST and completes execution of the program.

WHERE
displays the virtual address of a symbol or entry point, or the address of the next executable instruction. WHERE may also be used to display the module and CSECT name and the displacement into the CSECT corresponding to an address.

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:
TLC55.PAYER.LOAD(THRUST)
The program is a load module and is not a command processor.
The prefix in the user's profile is TLC55.
The parameters to be passed: 2048, 80

```
test payer(thrust) '2048,80'
```

or

```
test payer.load(thrust)
```

Example 3

Operation: Invoke a program for testing.

Known:
The name of the data set that contains the program: TLC55.PAYLOAD.OBJ
The prefix in the user's profile is TLC55.
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: TLC55.CMDS.LOAD(OUTPUT)

```
test cmds(output) cp
```

or

```
test cmds.load(output) cp
```

Note: You will be prompted to enter a command for the command processor. (TSO prompts you for the commands you wish to test.)

Example 5

Operation: Invoke a command processor for testing.

Known:
The name of the data set containing the command processor is TLC55.LOAD(OUTPUT).
The prefix in the user's profile is TLC55.

```
test (output) cp
```

Assignment of Values Function of TEST

When processing is halted at a breakpoint or before execution is initiated, you can modify values in virtual 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.

```
address = data-type 'value'
```

address
specifies the location that you want to contain a new value. Address can be:

- An absolute address
- A symbolic address
- A relative address
- An indirect address
- An address expression
- A module-name and entry-name (separated by a period)
- An entry-name (preceded by a period)
- A general register
- A floating point register

data-type 'value'[,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)[1]	Storage Boundary Data types must begin on specified boundary for a virtual storage address
C	Character	One line of input, continued lines permitted	C-byte
X	Hexadecimal	64	X-byte
B	Binary	64	B-byte
H	Fixed point binary (halfword)	6	H-halfword
F	Fixed point binary (fullword)	11	F-fullword
E	Floating point (single precision)	13	E-fullword
D	Floating point (double precision)	22	D-doubleword
P	Packed decimal	32	P-byte
Z	Zoned decimal	17	Z-byte
A	Address constant	11	A-fullword
S	Address (base + displacement)	8	S-halfword
Y	Address constant (halfword)	6	Y-halfword

[1] All characters within the quotes are included in the length.

Following is a list of valid entries and syntax for data type:

C 'character value'

X 'hexadecimal value'

B 'binary value'

H '[+] decimal value'
The minimum value for H-type is -32768 and the maximum value is 32767.

F '[+] decimal value'
The minimum value for F-type is -2147483648 and the maximum is 2147483647.

E '[+] decimal value [E[+] decimal exponent]'
A maximum of 8 digits are allowed for the decimal value and a maximum of 2 digits are allowed for the decimal exponent.

D '[+] decimal value [E[+] decimal exponent]'
A maximum of 17 digits are allowed for the decimal value and a maximum of 2 digits are allowed for the decimal exponent.

P '[+] decimal value'
A maximum of 31 digits are allowed.

Z '[+] decimal value'
A maximum of 16 digits are allowed.

A '[+] decimal value'
The minimum decimal value is -2147483648 and the maximum value is 2147483647.

S 'decimal value(register number)'
The decimal value can be from 0 to 4095 and the register number must be from 0 to 15 (decimal form).

Y '[+] decimal value'
The decimal value may be from 0 to 32767.

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, all other data types will be translated into uppercase (if necessary).

A list of data may be specified by enclosing the list in parentheses. The data in the list will be stored beginning at the location specified by the address operand.

Values assigned to general registers are placed in registers right-justified and padded with binary zeroes.

Notes:

- When a virtual storage address is assigned a list of data-type values the address must reside on the appropriate boundary for the specified data-type of the first value. Storage bytes for subsequent data-type values will be skipped to align data on the appropriate boundary for the data type requested.
- The following restrictions apply to general and floating-point registers:
- Only one data-type *should* be specified for floating-point registers. (Additional values are ignored.)

  - Assign only X or E data types to single precision floating-point registers.
  - Assign only X or D data types to double precision floating-point registers.
  - With the exception of the D-type of data, general registers can be assigned any data type.
- When a general register is assigned a list of data-type 'values', the first value is assigned to the specified register; subsequent data-type values are assigned to contiguous higher-numbered registers. If register 15 is reached and data-type values remain, the values are wrapped around to register 0 and subsequent registers if needed.
- If data is assigned to a storage area that contains a breakpoint, the breakpoint is removed and a warning message is displayed at the terminal.

Example 1

Operation: Insert a character string at a particular location in virtual 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'
```

Example 3

Operation: Initialize registers 0 through 3 to zeroes and register 15 to 4.

```
15R=(x'4',x'0',x'0',x'0'x'0')
```

Note: The sixteen (16) general registers are treated as contiguous fields with register 0 immediately following register 15.

Example 4

Operation: Assign a new base and displacement for an instruction that was found to be in error.

Known:
LA instruction at +30 is X'41309020'. In this instruction the current base register is 9 and the displacement is a decimal value of 32 (hexadecimal value of 20). The base register should be 10 and the decimal displacement should be 98 (hexadecimal value of 62).

```
+32=S'98(10)'
```

After this assignment the instruction at +30 will be:

```
X'4130A062'
```

Example 5

Operation: Insert a number in packed format at a particular address in virtual storage.

Known:
Absolute address: C3D41, decimal value to be packed is -1038.

```
c3d41.=p'-1038'
```

AT Subcommand of TEST

Use the AT subcommand to establish breakpoints where processing is to be temporarily halted so that you can examine the results of execution up to the point of interruption. Processing is halted before the instruction at the breakpoint is executed.

Note 1: A breakpoint should not be established at:

- The target of an execute instruction or the execute instruction itself
- An instruction that will be modified by the execution of other in-line code prior to the execution of the breakpoint
- An user written SVC exit

```
AT      { address[:address] }
        { (address-list)    }
        [(subcommands-list)]
        [COUNT(integer)]
        [NODEFER]
        [DEFER  ]
        [NOTIFY  ]
        [NONOTIFY]
```

---

address
specifies a location that is to contain a breakpoint. The address must be on a halfword boundary and contain a valid op code. See Note 2.

address:address
specifies a range of addresses that are to contain breakpoints. Each address must be on a halfword boundary. A breakpoint will be established at each instruction between the two addresses. When a range of addresses is specified, assignment of breakpoints halts when an invalid instruction is encountered. See Note 2.

address-list
specifies several addresses that are to contain breakpoints. Each 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 Note 2.

Note 2: Address can be:

- An absolute address
- A symbolic address
- A relative address
- An indirect address
- An address expression
- A module-name and entry name (separated by a period)
- An entry-name (preceded by a period)

subcommands-list
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. The list cannot be longer than 255 characters.

Note: If an OFF subcommand in the list removes the breakpoint for which a list is specified all remaining subcommands in that list are ignored.

COUNT(integer)
specifies that processing will not be halted at the breakpoint until it has been encountered the specified number of times. This operand is directly applicable to program loop situations, where an instruction is executed several times. Processing will be halted each time the breakpoint has been encountered for the number of times specified for the 'integer' operand. The integer specified cannot exceed 65,535.

DEFER
specifies that the breakpoint is to be established in a program that is **not yet** in virtual 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:

```
MODULENAME.ENTRYNAME.RELATIVE
```

or

```
MODULENAME.ENTRYNAME.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 virtual 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.

Note: If your program is running in supervisor state or in a PSW protection key less than 8, breakpoints are ignored.

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 that is not presently in virtual storage.

Known:
The name of the load module: CALCULAT
The CSECT name: INTEREST
The symbolic address for the breakpoint: TOTAL

```
at calculat.interest.total defer
```

Example 5

Operation: Have the following subcommands executed when the breakpoint at TAC is reached: LISTTCB PRINT(TCBS), LISTPSW, and GO CALCULAT

```
at tac (listtcb print(tcbs) listpsw;go calculat)
```

Example 6

Operation: Request that the following subcommands be executed when the breakpoint at symbol NOW is reached: LISTMAP, LISTTCB, OFF NOW, AT +32, and GO.

```
at now (listmap;listtcb;off now;at +32;go)
```

Note: The last two (2) subcommands will not be executed because the breakpoint (NOW) and its subcommand list will have been removed.

CALL Subcommand of TEST

Use the CALL subcommand to initiate processing at a specified address and to initialize registers 1, 14, and 15. You can pass parameters to the program that is to be tested.

Caution: The contents of registers 1, 14, and 15 are altered 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).

CALL **address**
[PARM(address-list)]
[VL]
[RETURN(address)]

---

address
specifies the address where processing is to begin. Register 15 contains this address when the program under test begins execution.

Address can be:

- An absolute address
- A symbolic address
- A relative address
- An indirect address
- An address expression
- A module-name and entry-name (separated by a period)
- An entry-name (preceded by a period)

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)
```

Note: The following message is issued after completion of the called routine:

```
'IKJ57023I PROGRAM UNDER TEST HAS TERMINATED NORMALLY+'
```

This message is issued because no return address was specified. If GO is now specified without an address, the TEST session will be terminated.

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)
```

Example 3

Operation: Initiate execution at label COMPUTE and have execution begin at label NEXT when control is returned via register 14.

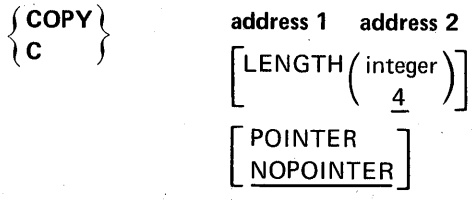
```
call compute return(next)
```

COPY Subcommand of TEST

Use the COPY subcommand to transfer data or addresses from one virtual storage address to another, from one general register to another, from a register to virtual storage, or from virtual 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.



address1
specifies a location that contains data to be copied.

address2
specifies a location that will receive the data after it is copied.

Address 1 and address 2 can be:

- An absolute address
- A symbolic address
- A relative address
- An indirect address
- An address expression
- A module-name and entry-name (separated by a period)
- A general register

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 maximum length is 256 bytes.

POINTER
specifies that address1 will be validity checked to see that it does not exceed maximum virtual storage 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.

NOPOINTER
specifies that address1 will be treated as an address. If neither POINTER nor NOPOINTER is specified, NOPOINTER is the default.

Note:

1. The COPY subcommand treats the 16 general registers as contiguous fields. The user can specify 10 bytes be moved from general register 0 to another location.

```
copy or 80060. length(10)
```

   The COPY subcommand will move the 4 bytes of register 0, the 4 bytes of register 1 and the high order 2 bytes of register 2 to virtual 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 two full words of data from one virtual 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. 1(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
```

Note: Specifying the pointer operand causes 80 to be treated as an immediate operand and not as an address.

Example 6

Operation: Insert the entry point of a routine into a virtual storage location.

Known:
The module name and the entry point name: IEFBR14.IEFBR14
The desired virtual storage location: STARTPTR

```
c iefbr14.iefbr14 startptr 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 real 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
```

DELETE Subcommand of TEST

Use the DELETE subcommand to delete, from virtual storage, a load module that was loaded by the tested program or one of its subtasks.

{DELETE / DEL} load-module-name

---

load-module-name
specifies the name of the load module to be deleted. The load name is the name (which might be and alias name) by which the program is known to the system when it is in virtual storage. The name must not exceed eight characters.

Example 1

Operation: Delete a load module from virtual storage.

Known:
The name of the load module: TOTAL

```
delete total
```

or

```
del total
```

DROP Subcommand of TEST

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 or EQUATE operand of the GETMAIN subcommand. You cannot remove symbols that were established by the linkage editor. If the program being tested was assembled with the TEST option and the EQUATE subcommand was used to override the location and type of the symbol within the program, then when the DROP subcommand is used to delete that symbol from the symbol table, the symbol will reflect the original location and type within the program.

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 subcommand.

```
drop
```

Example 2

Operation: Remove a symbol from the symbol table.

Known:
The name of the symbol is DATE.

```
drop date
```

Example 3

Operation: Remove several symbols from the symbol table.

Known:
The names of the symbols: STARTADD TOTAL WRITESUM

```
drop (startadd total writesum)
```

END Subcommand of TEST

Use the END subcommand to terminate all functions of the TEST command and the program being tested.

END

Note: The END subcommand will not close an opened data set; use the GO subcommand for this process. Normal exit cleanup procedures should also be used.

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. Symbol within DSECT may be accessed if the DSECT name is defined using the EQUATE subcommand. For restrictions on symbols see the section titled, "Internal Symbols". You can also modify the data attributes (type, length, and multiplicity); use the EQUATE subcommand to modify attributes of existing equated symbols. The DROP subcommand removes symbols added by the EQUATE subcommand. Symbols established via the EQUATE subcommand are defined for the duration of the TEST session only.

{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
the address that you specify will be equated to the symbol that you specify.

Address can be:

- An absolute address
- A symbolic address
- A relative address
- An indirect address
- An address expression
- A module-name and entry-name (separated by a period)
- An entry name (preceded by a period)

data-type
specifies 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
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

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.

Notes:

- If you do not specify any keywords, the defaults are:

```
type - X
multiplicity - 1
length - 4
```

- If both multiplicity and length are specified for data-type I, the multiplicity is ignored.

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)
```

Example 3

Operation: Add the symbol NAMES to the symbol table to access a list of 6 names. Each name is 8 characters long.

Known:

The names are stored one after the other at relative address +12C.

```
equate names +12c 1(8) m(6) c
```

FREEMAIN Subcommand of TEST

Use the FREEMAIN subcommand to free a specified number of bytes of virtual storage.

integer
specifies the number of decimal bytes of virtual storage to be released.

address
this address is the location of the space to be freed and must be a multiple of 8 bytes.
The LISTMAP subcommand may be used to help locate previously acquired virtual storage.

Address can be:

- An absolute address
- A symbolic address
- A relative address
- An indirect address
- An address expression
- A module-name and entry-name (separated by a period)
- An entry name (preceded by a period)

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 virtual storage that was acquired previously 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)
```

Example 2

Operation: Free space in virtual storage that was obtained previously by a GETMAIN subcommand.

Known:
The size of the space is 100 decimal bytes. The address of the space to be freed is A4 (hexadecimal form) past the address in register 3. The space to be freed is in subpool 0.

```
freemain 100 3r%+A4
```

Example 3

Operation: Free subpool 127.

```
freemain 0 0 sp(127)
```

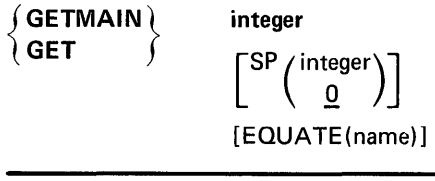
Warning: Do not attempt to free all of subpool 78. If you desire to free a portion of subpool 78, be careful not to free the storage obtained by the TMP. (This would result in your freeing the TMP's data areas because subpool 78 is shared.) The deletion of the TMP portion of subpool 78 will cause your session to terminate.

Note: You may release an entire subpool by specifying a length of 0, an absolute address of 0, and a subpool in the range of 1 through 127.

If you specify a non-zero address the length must also be non-zero.

GETMAIN Subcommand of TEST

Use the GETMAIN subcommand to obtain a specified number of bytes of virtual storage. The GETMAIN subcommand displays the starting address of the virtual storage obtained.



integer
specifies the number of decimal bytes, in decimal form, of virtual storage to be obtained.

SP(integer)
specifies the number of a subpool from which the virtual storage is to be obtained. If you omit this operand, the default value is subpool zero. The integer must be in the range zero through 127.

EQUATE(name)
specifies that the address of acquired virtual storage is to be equated to the symbol specified by "name". and placed in the TEST internal symbol table.

Example 1

Operation: Obtain 240 decimal bytes of virtual storage from subpool 0.

```
getmain 240
```

Example 2

Operation: Obtain 500 bytes of virtual 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.

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.

When the problem program completes processing, the following message is displayed at the terminal:

```
'IKJ57023I PROGRAM UNDER TEST HAS TERMINATED NORMALLY+'
```

If the GO subcommand is the issued with no address specified, the TEST session will be terminated.

Example 1

Operation: Begin execution of a program at the point where the last interruption occurred or initiate execution of a program.

```
go
```

Example 2

Operation: Begin execution at a particular address.

```
go calculat
```

HELP Subcommand of TEST

Use the HELP subcommand to obtain the syntax and function of the TEST subcommands. Refer to the HELP command for a description of the syntax and function of the HELP subcommand.

LIST Subcommand of TEST

Use the LIST subcommand to have the contents of a specified area of virtual storage, or the contents of registers, displayed at your terminal or placed into a data set.

{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. See the following note.

address:address
specifies that you want the data located between the specified addresses displayed at your terminal or placed into a data set. See the following note.

(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. If the first address of a range is a register, the second address must also be the same type of register (floating point or general). 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 the following note.

Note: Address can be:

- An absolute address
- A symbolic address
- A relative address
- An indirect address
- An address expression
- A module-name and entry-name (separated by a period)
- An entry name (preceded by a period)
- A general register
- A floating point register

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
P	Packed decimal	16
Z	Zoned decimal	16
A	Address constant	4
S	Address (base + displacement)	2
Y	Address constant (halfword)	2

All accepted data types will allow the specified address to be aligned on a byte boundary even though certain data types cannot be assigned to a byte boundary. The default for data-type is hexadecimal.

Notes:

1. A general register will be displayed in decimal format if the F data type is used. Otherwise, regardless of the type specified, a general register will be displayed in hexadecimal. Floating-point registers will be listed in floating-point format if data-type is not specified. However, floating-point registers can be listed in hexadecimal format by using the X data type. If any data type other than D, E, or X is specified for floating-point registers, data-type is ignored and the register is listed in floating-point format.
2. If an area is to be displayed using the I data type and that area contains an invalid op code, only the area up to that invalid op code will be displayed.
3. If a range of addresses is specified on LIST and the ending address is in fetch protected storage, the user will be prompted (if in PROMPT mode) to reenter the address. If a range of addresses is still desired, the user must reenter the range, not just the ending address.

LENGTH(integer)
indicates the length, in bytes of the data that is to be listed. If you use a symbolic address and do not specify LENGTH, the value for the LENGTH parameter will be retrieved from the internal TEST symbol table or from the length associated with a symbol in a program. 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

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 with the LENGTH operand. Gives the user the following options:

- The ability to format the data to be listed (see Example 7).
- A way of printing more than 256 bytes at a time. (The value supplied for "integer" determines how may "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 displayed. If you use a symbolic address and do not specify either LENGTH or MULTIPLE, the length retrieved from the internal TEST symbol table or from the program will be used and multiplicity will be ignored.

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 format and block size are treated as NEW if being opened for the first time, otherwise, they are treated as MOD data sets.
If PRINT(data-set-name) is specified, use the following table to determine the format of the output.
If the data-set-name is not specified within quotes, the descriptive qualifier, "TESTLIST" is added.

If your record type 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 those specified in the preceding table are unchanged.

The specified data set is kept open until:

- The TEST session is ended by a RUN or END subcommand, or
- 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 floating-point register 2 in single precision.

```
list  2e
```

Example 2

Operation: List all of the general registers.

```
list  0r:15r
```

Example 3

Operation: List all of the floating point registers in double precision.

```
list  0d:6d
```

Example 4

Operation: List 20 instructions starting with address +3A

```
list +3a i m(20)
```

Example 5

Operation: List the contents of an area of virtual storage.

Known:
The area to be displayed is between labels COUNTERA and DTABLE.
The data is to be listed in character format for a length of 130 bytes.
The name of the data set which the data is to be put is:
MYDATA.DCDUMP.

```
list countera:dtable
c 1(130) m(1) print ('mydata.dcdump')
```

Example 6

Operation: List the contents of virtual storage at several addresses.

Known:
The addresses: TOTAL1, TOTAL2, TOTAL3, and ALLTOTAL
Each address is to be displayed in fixed-point binary format in 3 lines of 3 bytes each.

```
list (total1 total2 total3 alltotal) f 1(3) m(3)
```

Example 7

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 data is to be listed in hexadecimal form in six lines of 4 bytes each.

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 display at your terminal will resemble the following:

```
+0      00000000
+4      00012A34
+8      00000B2C
+C      00000000
+10     001A0408
+14     00004430
```

Note: In the preceding example the hexadecimal data-type was not specified, it was the default.

LISTDCB Subcommand of TEST

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.

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.

LISTDCB **address**
[FIELD(names)]
[PRINT(data-set-name)]

---

address
specifies the address of the DCB that you want displayed. The address must be on a fullword boundary.

Address can be:

- An absolute address
- A symbolic address
- A relative address
- An indirect address
- An address expression
- A module-name and entry-name (separated by a period)
- An entry-name (preceded by a period)

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.

Following is a list of the valid field names for the DCB:

DCBBFALN	DCBFDAD
DCBBFTEK	DCBHIARC
DCBBUFCB	DCBIFLGS
DCBBUFL	DCBIOBAD
DCBBUFNO	DCBKEYCN
DCBDDNAM	DCBKLYLE
DCBDEBAD	DCBMACRF
DCBDEVT	DCBOFLGS
DCBBUFNO	DCBRECFM
DCBDVTBL	DCBRELAD
DCBLODAD	DCBTIOT
DCBEXLST	DCBTRBAL

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.

If the data-set-name is not specified within quotes, the descriptive qualifier TESTLIST is added.

If PRINT(data-set-name) is specified, use the following table to determine the format of the output.

If your record type was:	Fixed, Fixed Blocked, or Undefined		Variable or Variable Blocked	
Then it is changed to variable blocked with the following attributes:	Recordsize 125	Blocksize 1629	Recordsize 125	Blocksize 129

Note: Record and block sizes greater than those specified in the preceding table are unchanged.

The specified data set is kept open until:

- The LIST session is ended by a RUN or END subcommand, or
- 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 an entire DCB.

Known:
The absolute address of the DCB: A33B4

```
listdcb a33b4.
```

LISTDEB Subcommand of TEST

Use the LISTDEB subcommand to list the contents of a data extent block (DEB). You must provide the address of the DEB.

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.

LISTDEB **address**
[FIELD(names)]
[PRINT(data-set-name)]

---

address
specifies the address is the beginning of the DEB, and must be on a fullword boundary.

Address can be:

- An absolute address
- A symbolic address
- A relative address
- An indirect address
- An address expression
- A module-name and entry-name (separated by a period)
- An entry-name (preceded by a period)

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.

Following is a list of DEB names that are valid for the LISTDEB subcommand:

DEBAMLNG	DEBNMEXT
DEBAPPAD	DEBNMSUB
DEBDCBAD	DEBOFLGS
DEBDEBAD	DEBOPATB
DEBDEBID	DEBPRIOR
DEBECBAD	DEBPROTG
DEBEXSCL	DEBQSCNT
DEBFLGS1	DEBTCBAD
DEBIRBAD	DEBUSPRG

Following is a list of the valid DEB names in the direct access section:

DEBBINUM	DEBNMTRK
DEBDVMOD	DEBSTRCC
DEBENDCC	DEBSTRHH
DEBENDHH	DEBUCBAD

Note: These fields cannot be accessed unless there is a direct access section in the DEB.

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.

If the data-set-name is not specified within quotes, the descriptive qualifier TESTLIST is added.

If PRINT(data-set-name) is specified, use the following table to determine the format of the output.

If your record type was:	Fixed, Fixed Blocked, or Undefined		Variable or Variable Blocked	
Then it is changed to variable blocked with the following attributes:	Recordsize 125	Blocksize 1629	Recordsize 125	Blocksize 129

Note: Record and block sizes greater than those specified in the preceding table are 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 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 is 44 decimal (2C hexadecimal) bytes past the beginning of the DCB.
The address of the DEB: DCBIN+2C%

```
listdeb dcbin+2c%
```

Example 2

Operation: List the following fields in the DEB: DEBDCBAD and DEBOFLGS

Known:
The address of the DEB is 44 decimal (2C hexadecimal) bytes past the beginning of the DCB. The address of the DCB is in register 8.

```
listdeb 8r%+2c% field(debdcbad,deboflgs)
```

LISTMAP Subcommand of TEST

Use the LISTMAP subcommand to display a virtual storage map at the 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.

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 the 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.
If the data-set-name is not specified within quotes, the descriptive qualifier, TESTLIST, is added.
If PRINT(data-set-name) is specified, use the following table to determine the format of the output.

If your record type was:	Fixed, Fixed Blocked, or Undefined		Variable or Variable Blocked	
Then it is changed to variable blocked with the following attributes:	Recordsize 125	Blocksize 1629	Recordsize 125	Blocksize 129

Note: Record and block sizes greater than those specified in the preceding table are unchanged.

The specified data set is kept open until:

- The TEST session is ended by a RUN or END subcommand, or
- 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 virtual storage at your terminal.

```
listmap
```

Example 2

Operation: Direct a map of virtual storage to a data set.

Known:
The name for the data set: ACDQP.MAP.TESTLIST
The prefix in the user's profile is ACDQD.

```
listmap print(map)
```

LISTPSW Subcommand of TEST

Use the LISTPSW subcommand to display a program status word (PSW) at your terminal.

LISTPSW [ADDR(address)]
[PRINT(data-set-name)]

ADDR(address)
specifies 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.

Address can be:

- An absolute address
- A symbolic address
- A relative address
- An indirect address
- An address expression
- A module-name and entry-name (separated by a period)
- An entry-name (preceded by a period)

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.
If the data-set-name is not specified within quotes, the descriptive qualifier, TESTLIST, is added.
If PRINT(data-set-name) is specified, use the following table to determine the format of the output.

If your record type was:	Fixed, Fixed Blocked, or Undefined		Variable or Variable Blocked	
Then it is changed to variable blocked with the following attributes:	Recordsize 125	Blocksize 1629	Recordsize 125	Blocksize 129

Note: Record and block sizes greater than those specified in the preceding table are unchanged.

The specified data set is kept open until:

- The TEST session is ended by a RUN or END subcommand, or
- 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: Direct the input/output old PSW into a data set.

Known:
The prefix in the user's profile is ANZAL2.
The address of the PSW (in hexadecimal): 38.
The name for the data set: ANZAL2.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.

If you wish, you can have only selected fields displayed.

LISTTCB [ADDR(address)]
[FIELD(names)]
[PRINT(data-set-name)]

ADDR(address)
specifies 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.

Address can be:

- An absolute address
- A symbolic address
- A relative address
- An indirect address
- An address expression
- A module-name and entry-name (separated by a period)
- An entry-name (preceded by a period)

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.

Following is a list of the valid LISTTCB field names:

TCBABCUR	TCBIOBRC	TCBRBP
TCBAECB	TCBIOTIM	TCBRCMP
TCBAFFN	TCBIQE	TCBRTM12
TCBAQE	TCBJLB	TCBRTWA
TCBBACK	TCBJPQ	TCBSTABB
TCBCCPVI	TCBJSCB	TCBSTMCT
TCBCMP	TCBJSTCB	TCBSTPCT
TCBDAR	TCBLLS	TCBSWA
TCBDDEXC	TCBLMP	TCBSYSCT
TCBDDWTC	TCBLTC	TCBTCB
TCBDEB	TCBMSS	TCBTCBID
TCBDSP	TCBNDSP0	TCBTCT
TCBECB	TCBNDSP1	TCBTFLG
TCBESTAE	TCBNDSP2	TCBTID
TCBEXT1	TCBNDSP3	TCBTIO
TCBEXT2	TCBNDSP4	TCBTIRB
TCBFBYT1	TCBNDSP5	TCBTME
TCBFLGS	TCBNSTAE	TCBTMSAV
TCBFLGS6	TCBNTC	TCBTRN
TCBFLGS7	TCBOTC	TCBTSDP
TCBFOE	TCBPIE	TCBTSFLG
TCBFSAB	TCBPKE	TCBTSLP
TCBGRS	TCBPQE	TCBUSER
TCBGTFA	TCBQEL	

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.

If data-set-name is not specified within quotes, the descriptive qualifier, TESTLIST, is added.

If PRINT (data-set-name) is specified, use the following table to determine the format of the output.

If your record type was:	Fixed, Fixed Blocked, or Undefined		Variable or Variable Blocked	
Then it is changed to variable blocked with the following attributes:	Recordsize 125	Blocksize 1629	Recordsize 125	Blocksize 129

Note: Record and block sizes greater than those specified in the preceding table are unchanged.

The specified data set is kept open until:

1. The TEST session is ended by a RUN or a 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: Direct a copy of the TCB for the current task into a data set.

Known:
The prefix in the user's profile is NAN75.
The name of the data set: NAN75.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 of the data set: SCOTT.TCBDATA

```
listtcb addr(mytcb2) field(tcbtio,tcbcmp,tcbgrs)-
print('scott.tcbdata')
```

Example 3

Operation: List the entire TCB for the current task.

```
listtcb
```

LOAD Subcommand of TEST

Use the LOAD subcommand to load a program into real storage for execution.

LOAD	data-set-name

data-set-name
specifies the name of the partitioned data set containing the module to be loaded. Note that if the member name is not specified, TEMPNAME will be used. If the data-set-name is not specified within quotes the "LOAD" qualifier will be added.

Example 1

Operation: Load a program named GSCORES from the data set ATX03.LOAD.

Known:
The prefix in the user's profile is ATX03.

```
load 'atx03.load (gsores)'
```

or

```
load (gscores)
```

Example 2

Operation: Load a module named ATTEMPT from data set ATX03.TEST.LOAD.

Known:
The prefix in the user's profile is ATX03.

```
load 'atx03.test.load(attempt)'
```

or

```
load test(attempt)
```

However do not specify:

```
test.load(attempt)
```

as this results in ATX03.TEST.load.load being searched for.

Example 3

Operation: Load a module named PERFORM from data set ATX03.TRY.

```
load 'atx03.try(perform)'
```

OFF Subcommand of TEST

Use the OFF subcommand to remove breakpoints from a program.

OFF [address[:address]] [(address-list)]

---

address
specifies the location of a breakpoint that you want to remove. The address must be on a halfword boundary.

If no address is specified, all breakpoints are removed. Address can be:

- An absolute address
- A symbolic address
- A relative address
- An indirect address
- An address expression
- A module-name and entry-name (separated by a period)
- An entry-name (preceded by a period)

address:address
specifies a range of addresses. All breakpoints in the range of addresses will be removed. See the description of address for a list of valid address types.

(address-list)
specifies the location of several breakpoints that you want to remove. See the description of address for a list of valid address types.

Note: The list *must* be in parentheses with address separated by one or more blanks or a comma.

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 +2c 3r%

```
off (countra +2c 3r%)
```

Example 3

Operation: Remove all breakpoints in a program.

```
off
```

Example 4

Operation: Remove one (1) breakpoint.

Known:
The address of the breakpoint is in register 6.

```
off 6r%
```

QUALIFY Subcommand of TEST

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.

Alternately, you can specify an address to be used as the base location only for subsequent relative addresses. Each time you use the QUALIFY subcommand, previous qualifications are voided. Automatic qualification overrides previous qualifications via the QUALIFY subcommand. See the subsection titled "Qualified Addresses" at the beginning of this section for a more detailed description of qualified addresses.

Symbols that were established by the EQUATE subcommand before you enter QUALIFY are not affected by the QUALIFY subcommand.

```
{QUALIFY}    {address                                        }
{Q       }    {module-name[.entryname]  [TCB(address)]}
```

---

address
specifies the base location to be used in determining the absolute address for relative addresses only. It does not affect symbolic addressing. Address can be:

- An absolute address
- A symbolic address
- A relative address
- An indirect address
- An address expression
- A module-name and entry-name (separated by a period)
- An entry-name (preceded by a period)

module-name [entryname][TCB (address)]
specifies the name by which a load module is known, and optionally an externally referable name within a module. If only a module is specified, the first entry point in the module will be supplied.

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.

Note: When using the QUALIFY and WHERE (with relative addressing) command combination for routines such as user exit routines and validity check routines, the load module or CSECT indicated may differ from the one that was qualified. This is due to system control processing of automatic qualification.

Example 1

Operation: Establish the absolute address 5F820 as a base location for relative addressing.

```
qualify 5f820
```

Note: This is useful in referring to relative addresses (offsets) within a control block or data area.

Example 2

Operation: Establish a base location for resolving relative addresses.

Known:
The module name is BILLS.
The relative address is +2A.

```
qualify bills +2a
```

Example 3

Operation: Establish an address as a base location for resolving relative addresses.

Known:
The address is 8 bytes past the address in register 7.

```
q 7r%+8
```

Example 4

Operation: Establish a base location for relative addresses to a symbol within the currently qualified program.

Known:
The base address: QSTART

```
qualify qstart
```

Example 5

Operation: Establish a symbol as a base location for resolving relative addresses.

Known:
The module name is MEMBERS
The CSECT name is BILLS.
The symbol is NAMES.

```
qualify members.bills.names
```

Example 6

Operation: Define the base location for relative and symbolic addressing.

Known:
The base location is the address of a program named OUTPUT.

```
q output
```

Example 7

Operation: Change the currently qualified module and CSECT. This means defining the base location for relative and symbolic addresses to a new program. The module can be a unique name under any task, or a module under the current task (where there is another one by the same name under a different task, the module under the current task would be used).

Known:
The module name is PROFITS.
The CSECT name is SALES.

```
qualify profits.sales
```

Example 8

Operation: Change the base location for symbolic and relative addresses to a module that has an identical name as another module under a different task.

Known:
The module name is SALESRPT.
The desired module is the one under the task represented by the TCB whose address is in general register 5.

```
q salesrpt tcb(5r%)
```

RUN Subcommand of TEST

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.

{RUN / R} [address]

---

address
execution begins at the specified address. If you do not specify an address, execution begins at the last point of interruption or at the entry point if the GO or CALL subcommand was not previously specified.

Address can be:

- An absolute address
- A symbolic address
- A relative address
- An indirect address
- An address expression
- A module-name and entry-name (separated by a period)
- An entry-name (preceded by a period)

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
```

WHERE Subcommand of TEST

Use the WHERE subcommand to obtain an absolute address, the name of a module and CSECT, a relative offset within the CSECT, and the address of the TCB for the specified address. You may also 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 entry point 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.

{WHERE / W} {address / module-name}

---

address
Address can be:

- An absolute address
- A symbolic address
- A relative address
- An indirect address
- An address expression
- A module-name and entry-name (separated by a period)
- An entry-name (preceded by a period)

module-name
specifies the name by which a load module is known or the name of an object module. The output of the WHERE subcommand is the module name, the CSECT name, the offset within the CSECT, the absolute address, and the address of the TCB. If only the module name was specified, the only output will be the absolute address of the module and the address of the TCB for the task under which the module was found. If the specified address is *not* within the extent of any user program, only the absolute address is returned. (Along with the absolute address a message will be returned stating that the specified address is not within the program extent.) If no operands are specified, the absolute address returned is the address of the next executable instruction.

Example 1

Operation: Determine the absolute address of the next executable instruction.

```
where
```

Example 2

Operation: Determine in which module an absolute address is located.

Known:
The absolute address: 3E2B8

```
where 3e2b8.
```

Example 3

Operation: Obtain absolute address of +2c4.

```
w +2c4
```

Note: An unqualified relative address is calculated from the currently qualified address (as specified via the QUALIFY command or the current module and CSECT, if no other qualification exists). The module name, CSECT name and TCB address are also obtained along with the absolute address.

Example 4

Operation: Obtain offset of the symbol SALES in the current program.

```
where   sales
```

Note: The module name, CSECT name, absolute address, and the TCB address are returned along with the offset of SALES.

Example 5

Operation: Determine in which module the address in register 7 is located.

```
w   7r%
```

Note: The offset, absolute address, and the TCB address are also returned with the module name.

Example 6

Operation: Obtain the virtual address of the module named CSTART.

```
where cstart
```

Example 7

Operation: Obtain the virtual address of the CSECT named JULY in the module named NETSALES.

```
where netsales.july
```

Example 8

Operation: Determine the relative address of symbol COMPARE in the module named CALCULAT and CSECT named AVERAGE.

```
w calculat.average.compare
```

Note: The absolute address and TCB address are also returned with the relative address.

Example 9

Operation: Determine the virtual address of +1CA.

Known:
The CSECT is MARCH.
The module is GETDATA.

```
where getdata.march.+1ca
```

Note: You will also get the TCB address with the virtual address.

Example 10

Operation: Obtain the absolute address for relative address +2C in CSECT named PRINTIT within the currently qualified module.

```
where .printit.+2C
```

TIME Command

Use the TIME command to obtain the following information:

- Cumulative CPU time (from LOGON)
- Cumulative session time (from LOGON)
- Service units used

  where *service units* can be:

  CPU Service Units - The task execution time, divided by an SRM constant, that is CPU model-dependent.

  I/O Service Units - The sum of individual SMF data set activity EXCP counts for all data sets associated with the address space.

  Storage Service Units - The number of real page frames multiplied by CPU service units, multiplied by .02. The decimal .02 is a scaling factor designed to bring the storage service component in line with the CPU component.

- Local time of day

  where "local time of day" refers to the time of execution for this command. It is displayed in the following manner:

```
local time of day in hours(HH),
minutes(MM), and seconds(SS),
(am or pm is also displayed)
```

- Today's date

To enter the command while a program is executing, you must first cause an attention interruption. The TIME command has no effect upon the executing program.

TIME

WHEN Command

Use the WHEN command 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.

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 numeric 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-name
specifies any valid TSO command name and appropriate operands. The command will be processed if the comparison is true.

Note 1: WHEN will terminate CLIST processing and then execute the TSO command-name specified.

Note 2: Successive WHEN commands may be used to determine an exact return code and then perform some action based on that return code.

Example 1: Using successive WHEN commands to determine an exact return code.

```
CALL        compiler
WHEN        SYSRC(= 0) EXEC LNKED

WHEN        SYSRC(= 4) EXEC LNKED

WHEN        SYSRC(= 8) EXEC ERROR
```

Command Procedures

A command procedure is a prearranged executable sequence of TSO commands, subcommands, and command procedure statements that can be invoked by issuing the EXEC command or the EXEC subcommand of EDIT. It is often referred to as a CLIST (command list). The TSO commands and subcommands have been described in previous sections of this book. This section describes command procedure statements and functions that can be used with them.

You should be familiar with the more detailed descriptions of command procedures found in *OS/VS2 TSO Terminal User's Guide*. This section is intended to be reference material and does not deal with all aspects of the use of command procedures.

See the description of the EXEC command in this book for information on invoking command procedures.

Functions Available for Command Procedures

The facilities that can be used with command procedure statements are:

- Symbolic variables, which can be specified on TSO commands, subcommands, and command procedure statements
- Control variables, a type of symbolic variable, which can only be specified on command procedure statements
- Built-in functions, a type of symbolic variable, which can only be specified on command procedure statements

An expression consists of these variables, whole numbers, and character strings combined with operators. Expressions are used on some of the command procedure statements.

The following topics describe the use of expressions and operators, symbolic variables, control variables, and built-in functions.

Figure 13 is a coding reference for command procedures. It lists, in alphabetic order, command procedure statements and facilities that can be coded on command procedure statements, a brief description of each, and the topic under which each is discussed.

Name	Meaning	See
< (or LT)	Less than	Expressions and Operators
<= (or LE)	Less than or equal	Expressions and Operators
+	Addition	Expressions and Operators
| (or OR)	Or	Expressions and Operators
& & (or AND)	And	Expressions and Operators
&DATATYPE	Determine expression type	Built-In Functions
&EVAL	Evaluate arithmetic expression	Built-In Functions
&LASTCC	Get last return code	Control Variables
&LENGTH	Determine expression length	Built-In Functions
&MAXCC	Get highest return code	Control Variables
&STR	Define character string	Built-In Functions
&SUBSTR	Define substring	Built-In Functions
&SYSDATE	Current date	Control Variables
&SYSDLM	Terminal delimiter	Control Variables
&SYSDVAL	Terminal delimiter parameters	Control Variables
&SYSICMD	Implicit execution member name	Control Variables
&SYSNEST	Nested procedure indicator	Control Variables
&SYSPCMD	Current primary command name	Control Variables
&SYSPREF	Data set name prefix	Control Variables
&SYSPROC	Logon procedure name	Control Variables
&SYSSCAN	Symbolic substitution scan limit	Control Variables
&SYSSCMD	Current subcommand name	Control Variables
&SYSTIME	Current time	Control Variables
&SYSUID	Current userid	Control Variables
*	Multiplication	Expressions and Operators
**	Exponentiation	Expressions and Operators
¬> (or NG)	Not greater than	Expressions and Operators
¬< (or NL)	Not less than	Expressions and Operators
¬= (or NE)	Not equal	Expressions and Operators
-	Subtraction	Expressions and Operators
/	Division	Expressions and Operators
//	Remainder	Expressions and Operators
> (or GT)	Greater than	Expressions and Operators
>= (or GE)	Greater than or equal	Expressions and Operators
= (or EQ)	Equal	Expressions and Operators
AND	And	Expressions and Operators
ATTN	Attention exit	Command Procedure Statements
CLOSFILE	Close open file	Command Procedure Statements
CONTROL	Control options	Command Procedure Statements
DATA(-ENDDATA)	Starts DATA group	Command Procedure Statements
DO(-WHILE-END)	Start DO group	Command Procedure Statements
(IF-THEN-)ELSE	Start IF-not action	Command Procedure Statements
(DO-WHILE-)END	End DO group	Command Procedure Statements

Figure 13. Command Procedure Coding Reference (Part 1 of 2)

Name	Meaning	See
END	End the command procedure	END Command
(DATA-)ENDDATA	Ends DATA group	Command Procedure Statements
EQ	Equal	Expressions and Operators
ERROR	Error exit	Command Procedure Statements
EXEC	Invoke a command procedure	EXEC Command
EXIT	Exit from nested procedure	Command Procedure Statements
GE	Greater than or equal	Expressions and Operators
GETFILE	Get record from open file	Command Procedure Statements
GLOBAL	Define global symbolic variables	Command Procedure Statements
GOTO	Unconditional branch	Command Procedure Statements
GT	Greater than	Expressions and Operators
IF(-THEN-ELSE)	Tests IF condition	Command Procedure Statements
LE	Less than or equal	Expressions and Operators
LT	Less than	Expressions and Operators
NE	Not equal	Expressions and Operators
NG	Not greater than	Expressions and Operators
NL	Not less than	Expressions and Operators
OPENFILE	Open a file	Command Procedure Statements
OR	Or	Expressions and Operators
PROC	Set and use symbolic parameters	Command Procedure Statements
PUTFILE	Put record into open file	Command Procedure Statements
READ	Get input from terminal	Command Procedure Statements
READDVAL	Get input from &SYSDVAL	Command Procedure Statements
RETURN	Return control from attn/err exit	Command Procedure Statements
SET	Assign values to variables	Command Procedure Statements
TERMIN	Request terminal input	Command Procedure Statements
(IF-)THEN(-ELSE)	Start IF action	Command Procedure Statements
WHEN	Inspect program return code	WHEN Command
(DO-)WHILE(-END)	DO loop control	Command Procedure Statements
WRITE	Send output to terminal	Command Procedure Statements
WRITENR	Send output to terminal with no return at end	Command Procedure Statements

Figure 13. Command Procedure Coding Reference (Part 2 of 2)

Expressions and Operators

Operators are used in command procedures to specify operations to be performed on terms in an expression. Operators are in three categories:

- Arithmetic operators, which specify fixed-point arithmetic operations to be performed on numeric operands. These operators connect whole numbers, character strings, symbolic variables, control variables, and built-in functions to form simple expressions.
- Comparative operators, which specify comparison functions to be performed between two simple expressions, and thereby form comparative expressions.
- Logical operators, which specify a logical connection between two comparative expressions, and thereby form logical expressions.

Figure 14 lists the operators in the three categories and shows how to enter them.

	For the function:	Enter:
Arithmetic	Addition	+
	Subtraction	-
	Multiplication	*
	Division	/
	Exponentiation	** (see Note 1)
	Remainder	//
Comparative	Equal	= or EQ
	Not equal	¬ = or NE
	Less than	< or LT
	Greater than	> or GT
	Less than or equal	< = or LE
	Greater than or equal	> = or GE
	Not greater than	¬ > or NG
	Not less than	¬ < or NL
Logical	And	& & or AND
	Or	| | or OR
Note 1: Negative Exponents are handled as exponents of zero.		

Figure 14. Arithmetic, Comparative, and Logical Operators

Symbolic Variables

The term "symbolic variable" refers to any character string in a command procedure for which different values may be substituted at different times. Symbolic variables add flexibility to command procedures by symbolizing real values that can change dynamically during execution of a command procedure and that can be different for each invocation of a command procedure.

A symbolic variable consists of an ampersand (&) followed by a maximum of 31 alphameric characters, the first of which is alphabetic. Types of symbolic variables are:

- Parameters on PROC, READ, or READDVAL statements
- Control variables
- Built-in functions
- Global variables on GLOBAL statements
- File names on OPENFILE, CLOSFILE, GETFILE, and PUTFILE statements

You define a symbolic variable by including it on a SET, GLOBAL, READ, READDVAL, PROC, or OPENFILE statement. Symbolic variables are replaced by real values during a process called symbolic substitution. Concatenation can be used to create new variables on SET and OPENFILE statements. The following topics describe symbolic substitution and concatenation.

You may use abbreviations of the symbolic variables as long as the abbreviation is not a duplication of any existing operand.

Symbolic Substitution

Symbolic substitution is the process of replacing symbolic variables with real values. Each line is scanned from left to right, and the symbolic variables are replaced with their real values. The real value substituted for a symbolic variable may actually be another symbolic variable (nested symbolic variables). If there are nested symbolic variables, the line is scanned more than once to resolve all symbolic variables. (You can limit the number of times a line can be rescanned by setting a control variable.)

The use of double ampersands requires special processing by the symbolic substitution routine. Each pair of ampersands is replaced by a single ampersand. This substitution takes place only after all other symbolic substitution in a line is complete. Consider the following:

```
set &a = &str( &&x)
```

After symbolic substitution, the value of & a is the string & x, which is another symbolic variable. An exception to this rule for substitution of double ampersands is the file name on a file I/O statement, in which case double ampersands are not replaced.

Concatenation of Symbolic Variables

Concatenation can be used to establish variables on SET and OPENFILE statements. Concatenation of symbolic variables consists of writing the symbolic variable names next to each other with no delimiters. For example:

```
&a&b&c
```

Concatenating symbolic variables and character strings requires use of a period as a delimiter when the symbolic variable precedes the character string. For example:

```
&varname.alpha
```

No delimiter is required when the character string precedes the symbolic variable. For example:

```
alpha&type
```

Character Set Supported in Command Procedure Variables

Using command procedure file I/O statements can cause characters other than those you can enter at a terminal to become part of the value of a symbolic variable. Certain hexadecimal codes are used by the system in command procedure internal processing and should not appear in data processed by command procedure file I/O statements. Command procedures support all codes from x'40' through x'FF', with the understanding that lowercase characters are translated to uppercase and lowercase numbers (x'B0'-x'B9') are translated to standard numbers (x'F0-x'F9'). Additionally, the following control characters are supported:

- x'05' HT (Horizontal tab)
- x'14' RES (Restore)
- x'16' BS (Backspace)
- x'17' IL (Idle)
- x'24' BYP (Bypass)
- x'25' LF (line feed)

All other codes between x'00' and x'3F' are reserved for command procedure internal processing; the use of file I/O statements to process data sets containing these codes is not supported. For example, file I/O statements cannot be used to process OBJ or LOAD type data sets.

Refer to *IBM System/370 Reference Summary* for the characters associated with the internal hexadecimal codes.

Control Variables

Control variables can be used in command procedures to obtain information about the current command procedure environment and the user who invoked the command procedure. To obtain and use this information, specify the appropriate symbolic variable in a command procedure statement. TSO replaces the symbolic variable with the current information.

Four of these control variables can be set or changed by the writer of the command procedure. These are & LASTCC, & MAXCC, & SYSDVAL, and & SYSSCAN. If the writer tries to change any of the other control variables, an error message is issued.

The control variables and their uses are described in Figure 15.

Symbolic Variable	Use	Can be changed by the writer
& LASTCC	To obtain the return code from the last operation, whether TSO command, subcommand, or command procedure statement. (See Note 1)	Yes
& MAXCC	To obtain the highest return code issued up to this point in the command procedure or passed back from a nested command procedure. The return code is in decimal format. (See Note 1.)	Yes
& SYSDATE	To obtain the present date in the format mm/dd/yy, where mm is month, dd is day, and yy is year.	No
& SYSDLM	To identify which character string, of those specified on the TERMIN statement, the terminal user entered to return control to the command procedure.	No
& SYSDVAL	(1) To obtain any parameters the terminal user entered, besides the delimiter, when he returned control to the command procedure after a TERMIN statement. (2) To obtain the terminal user's response line when a READ statement requests terminal input.	Yes
& SYSICMD	To obtain the name by which the user implicitly invoked this command procedure. This value is null if the command procedure was invoked explicitly.	No
& SYSNEST	To determine if the currently executing command procedure was invoked from another procedure. & SYSNEST is replaced with "YES" if this is a nested procedure and "NO" if it is not.	No
& SYSPCMD	To obtain the name (or abbreviation) of the most recently executed TSO command (with the exception of the TIME command) in this procedure. The initial value is "EXEC" (or "EX") in the command environment and "EDIT" (or "E") in the subcommand environment.	No
& SYSPREF	To obtain the data-set-name prefix from the user profile table (UPT) for the command procedure user.	No
& SYSPROC	To obtain the procedure name specified when the command procedure user logged on.	No
& SYSSCAN	To obtain the maximum number of times that symbolic substitution is allowed to rescan a line to evaluate symbolic variables. The default is 16 times. The maximum value is two to the 31st power minus one (+2, 147, 483, 647); the minimum is 0.	Yes
& SYSSCMD	To obtain the name (or abbreviation) of the subcommand currently executing. The initial value is null if EXEC was issued in the command environment and 'EXEC' (or 'EX') if EXEC was issued as a subcommand of EDIT. The value is null whenever the procedure is in the command environment.	No
& SYSTIME	To obtain the present time in the format hh:mm:ss, where hh is hours, mm is minutes, and ss is seconds.	No
& SYSUID	To obtain the userid of the user currently executing the command procedure.	No

Note 1: The command procedure statement return codes are in Figure 18. The TSO command and subcommand return codes are:

0 Normal completion.

12 A terminating error occurred during execution; however, the command processor might have been able to prompt for information necessary to recover from the error.

Figure 15. Control Variables

Built-In Functions

Built-in functions can be used in command procedures to perform certain evaluations of expressions and character strings. To request a built-in function, specify the appropriate symbolic variable with an expression or character string on a command procedure statement. TSO evaluates the expression first, if necessary, and then performs the requested function. The symbolic variable is replaced by the result of performing the built-in function.

The built-in functions are &DATATYPE, &LENGTH, &EVAL, &STR, and &SUBSTR. Their uses are described in Figure 16.

Symbolic variable	Use
&DATATYPE(expression)	To find out whether an evaluated expression is entirely numeric. &DATATYPE is replaced by 'NUM' if the expression is all numeric or by 'CHAR' if there is at least one non-numeric character.
&EVAL(expression)	To find the result of an arithmetic expression. &EVAL is replaced by the result of evaluating the expression.
&LENGTH(expression)	To find the number of characters in the result of an evaluated expression. &LENGTH is replaced by the number of characters in the result. (Leading zeroes are ignored.)
&STR(string)	To use the indicated string as a real value. Nested built-in functions and symbolic substitution are performed but no other evaluation is done. &STR is replaced by the string.
&SUBSTR(expression[:expression],string)	To use the indicated portion of a string as a real value. Nested built-in functions and symbolic substitution are performed but no other evaluation is done. &SUBSTR is replaced by the specified portion of the string (substring). The start and end of the substring are indicated by the two expressions. To select a one-character substring, you need to enter only the first expression.

Figure 16. Built-In Functions

Command Procedure Statements

Command procedure statements assign values, set controls, select options, and control the conditions under which command procedures execute. Statements operate in both the command and subcommand environment, which means that statements will work in command procedures invoked either by the EXEC command or by the EXEC subcommand of EDIT. In general, statements fall into control, assignment, conditional, and file access categories. See Figure 17.

Control	Assignment	Conditional	File Access
ATTN	READ	DO-WHILE-END	CLOSFILE
CONTROL	READDVAL	IF-THEN-ELSE	GETFILE
DATA-ENDDATA	SET	(WHEN Command)	OPENFILE
ERROR			PUTFILE
EXIT			
GLOBAL			
GOTO			
PROC			
RETURN			
TERMIN			
WRITE			
WRITENR			

Figure 17. Command Procedure Statement Categories

Figure 18 lists the error codes set by the command procedure statements.

16	Not enough virtual storage
300	User tried to update an unauthorized variable
304	Invalid keyword on EXIT statement
308	Code specified, but no code given on EXIT statement
312	Internal GLOBAL processing error
316	TERMIN delimiter greater than 256 characters
324	GETLINE error
328	More than 64 delimiters on TERMIN
332	Invalid file name syntax
336	File already open
340	Invalid OPEN type syntax
344	Underfined OPEN type
348	File specified did not open (for example, the filename was not allocated)
352	GETFILE - filename not currently open
356	GETFILE - the file has been closed by the system (for example, file opened under EDIT and EDIT has ended)
360	PUTFILE - file name not currently open
364	PUTFILE - file closed by system (see code 356)
368	PUTFILE - CLOSFILE - file not opened by OPENFILE
372	PUTFILE - issued before GETFILE on a file opened for update
400	GETFILE end of file (treated as an error, which can be handled by ERROR action)
8xx	Evaluation routine error codes
800	Data found where operator was expected
804	Operator found where data was expected
808	A comparison operator was used in a SET statement
812	(Reserved)
816	Operator found at the end of a statement
820	Operators out of order
824	More than one exclusive operator found
828	More than one exclusive comparison operator
832	The result of an arithmetical calculation is outside the range extending from -2,147,483,684 to +2,147,483,647.

Figure 18. Command Procedure Statement Error Codes (Decimal) (Part 1 of 2)

	836	(Reserved)
	840	Not enough operands
	844	No valid operators
	848	Attempt to load character from numeric value
	852	Addition error - character data
	856	Subtraction error - character data
	860	Multiplication error - character data
	864	Divide error - character data or division by 0
	868	Prefix found on character data
	872	Numeric value too large
	900	Single ampersand found
	904	Symbolic variable not found
	908	Error occurred in an error action range that received control because of another error
	912	Substring range invalid
	916	Non-numeric value in substring range
	920	Substring range value too small (zero or negative)
	924	Invalid substring syntax
	932	Substring outside of the range of the string, for example, 1:3,AB; (AB is only two characters)
	936	A built-in function that requires a value was entered without a value
	940	Invalid symbolic variable
	944	A label was used as a symbolic variable
	948	Invalid label syntax on a GOTO statement
	952	GOTO label was not defined
	956	GOTO statement has no label
	960	&SYSSCAN was set to an invalid value
	964	&LASTCC was set to an invalid value and EXIT tried to use it as a default value
	968	DATA PROMPT-ENDDATA statements supplied, but no prompt occurred.
	972	TERMIN command cannot be used in background jobs.
	999	Internal command procedure error
*	Sxxx	A system ABEND code
*	Uxxx	A user ABEND code

* Printed in hexadecimal

Figure 18. Command Procedure Statement Error Codes (Decimal) (Part 2 of 2)

ATTN Statement

The ATTN statement sets up an environment that detects attention interruptions processed by the terminal monitor program (TMP). The detection of an attention interruption invokes a specified action which is considered to be an attention exit.

[label:] **ATTN** [OFF / action]

---

label:
specifies a name to which the command procedure can branch. Enter one-to-eight characters, the first alphabetic and the rest alphameric, followed by a colon and at least one blank.

OFF
specifies that any previous attention action is nullified. When no action is specified on the ATTN statement, OFF is the default.

action
specifies any executable statement, commonly a DO-group constituting a routine. This routine must specify either a command or a null before the RETURN statement. Results:

Null: Ignore the attention.

Not-null (a command was specified): Give control to the command that was specified.

Example

Operation: Pass control to a command on an attention exit.

```
ATTN    DO
        SET &CMD=     /* Default to null */
        WRITE ATTENTION IN CONTROL
        IF  &OKTOTERMINATE=YES THEN +
            DO
                WRITE DO YOU WANT TO TERMINATE (Y OR N)
                READ &ANS
                IF &ANS=Y THEN +
                    SET &CMD=END
            END
        ELSE +
            WRITE IGNORING YOUR ATTENTION
        &CMD            /* The TSO command */
        RETURN
END
.
.
```

CLOSFILE Statement

The CLOSFILE statement is used to close a file that was previously opened by an OPENFILE statement. It is not necessary to specify file type. Only one file can be closed with one statement.

File variables are only scanned once (no rescans) and only on OPENFILE.

[label:] **CLOSFILE filename**

---

label:
specifies a name to which the command procedure can branch. Enter one-to-eight characters, the first alphabetic and the rest alphameric, followed by a colon and at least one blank.

filename
specifies the ddname by which the file was allocated and opened (via OPENFILE).

The CONTROL statement defines certain processing options to be in effect for the command procedure. The options are in effect from the time CONTROL executes until either the procedure terminates or another CONTROL is issued.

Command procedures without CONTROL statements execute with options MSG, NOLIST, NOPROMPT, NOCONLIST, NOSYMLIST, and FLUSH. The user can set PROMPT and LIST by entering them as keywords on the EXEC command or subcommand that invokes the command procedure.

CONTROL has no default operands. If you enter CONTROL with no operands, the system uses options already in effect because of system predefinition, presetting via EXEC, or setting by a previous CONTROL statement. In addition, when there are no operands specified, the system will display those options which are currently in effect.

Note: CONTROL operands cannot be entered as symbolic variables.

```
[label:] CONTROL [FLUSH   ]
                 [NOFLUSH ]
                 [PROMPT   ]
                 [NOPROMPT ]
                 [LIST     ]
                 [NOLIST   ]
                 [CONLIST  ]
                 [NOCONLIST]
                 [SYMLIST  ]
                 [NOSYMLIST]
                 [MSG      ]
                 [NOMSG    ]
                 [MAIN]
                 [END(string)]
```

---

label:
specifies a name to which the command procedure can branch. Enter one-to-eight characters, the first alphabetic and the rest alphameric, followed by a colon and at least one blank.

FLUSH
specifies that the system can purge (flush) the queue called the input stack. The system normally flushes the stack on an execution error.

NOFLUSH
specifies that the system cannot flush the stack.

PROMPT
specifies that the command procedure can prompt the terminal for input.

NOPROMPT
specifies that the command procedure cannot prompt the terminal for input, even if the procedure has prompting capabilities.

LIST
specifies that commands and subcommands are displayed at the terminal after symbolic substitution but before execution.

NOLIST
specifies that commands and subcommands are not displayed at the terminal after symbolic substitution but before execution.

CONLIST
specifies that command procedure statements are displayed at the terminal after symbolic substitution but before execution.

NOCONLIST
specifies that command procedure statements are not displayed at the terminal after symbolic substitution but before execution.

SYMLIST
specifies that executable statements are displayed at the terminal once before the scan for symbolic substitution. Executable statements include commands, subcommands, and command procedure statements.

NOSYMLIST
specifies that executable statements are not displayed at the terminal before symbolic substitution.

MSG
specifies that PUTLINE informational messages from commands and statements in the procedure are displayed at the terminal.

NOMSG
specifies that PUTLINE informational messages NOMSG from commands and statements in the command procedure are not displayed at the terminal.

MAIN
specifies that this is the main command procedure in your TSO environment and cannot be deleted by a stack flush request from the system. When MAIN is specified, FLUSH and NOFLUSH are ignored. The attention exit in the TMP cannot delete the command procedure and any error exit used by this command procedure is protected.

END(string)
specifies that a character string will be recognized by the system as an END statement that concludes a DO-group. Enter the string as 1-4 characters, the first alphabetic and the rest alphameric. Since END no longer signifies the end of a DO-group, the writer of the command procedure can include END commands and subcommands without prematurely ending the DO-group.

DATA-ENDDATA Sequence

The DATA and ENDDATA statements are used to designate a group of commands and subcommands that are looked at as data by the command procedure but as commands and subcommands by the system. Symbolic substitution is performed before execution of the group. Command procedure statements included in the DATA-ENDDATA group cause failures because TSO attempts to execute them as commands or subcommands. A DO-group ignores an END in an included DATA-ENDDATA group, instead of terminating the DO-group.

```
[label:]        DATA
                  :
                  :
                ENDDATA
```

label:
specifies a name to which the command procedure can branch. Enter one-to-eight characters, the first alphabetic and the rest alphameric, followed by a colon and at least one blank. You cannot specify a label for ENDDATA.

Example

Operation: Perform an EDIT operation without ending a DO-group.

```
.
.
IF &ADDIT=YES THEN -
    DO
        DATA
            EDIT OLD.DATA
            BOTTOM
            INSERT * &NEW ENTRY
            END SAVE
        ENDDATA
        .
        .
    END
    .
    .
ELSE
.
.
```

DO-WHILE-END Sequence

The DO, WHILE and END statements are used to form commands, subcommands, and statements into DO-groups of related instructions. DO and END denote the start and end, respectively, of the DO-group. WHILE specifies a condition and causes the DO-group to re-execute as long as the condition is true.

The string specified on the END operand of the CONTROL statement can be used instead of the END statement.

```
[label:]        DO [WHILE logical-expression]
                   .
                   .
                   .
[label:]        END
```

label:
specifies a name to which the command procedure can branch. Enter one-to-eight characters, the first alphabetic and the rest alphameric, followed by a colon and at least one blank.

logical-expression
is a group of comparative expressions grouped by logical operators (see "Definitions of Command Procedure Terminology"). The minimal entry for logical-expression is a comparative expression.

ERROR Statement

The ERROR statement sets up an environment that checks for nonzero (error-condition) return codes from commands, subcommands, and command procedure statements in the currently executing command procedure. When an error code is detected, an action can be invoked. This action is effectively an error exit.

The error exit must be protected from being flushed from the input Stack by the system. Stack flushing makes the error return codes unavailable. Use the MAIN or NOFLUSH operands of the CONTROL statement to prevent stack flushing.

When ERROR is entered with no operands, the system displays any command, subcommand, or statement in the command procedure that ends in error. The system then attempts to continue with the next sequential statement, if possible.

Note: If the LIST option was requested for the command procedure being executed the NULL error statement will be ignored.

The ERROR statement must precede any statements that might cause a branch to it.

avoid an endless loop

[label:] **ERROR** [OFF | action]

---

label:
specifies a name to which the command procedure can branch. Enter one-to-eight characters, the first alphabetic and the rest alphameric, followed by a colon and at least one blank.

OFF
specifies that any action previously set up by an ERROR statement is nullified. Note that OFF is not a default.

action
specifies any executable statement, commonly a DO-group constituting a routine.

Example

Operation: Perform an error analysis routine whenever an error occurs in the command procedure.

```
.
.
ERROR   DO
        .
        .   /* Error analysis routine */
        .
        END
.
.
```

EXIT Statement

The EXIT statement causes control to be returned to the routine that called the currently executing command procedure. The return code associated with this exit can be specified by the user or allowed to default to the value in control variable & LASTCC.

A procedure that is called by another procedure is said to be nested. A called procedure can also call a procedure, which would be considered to be nested two levels. Levels of nesting are limited only by the extent of storage and the skill of the programmer. The structure of the nesting is called the hierarachy. You go "up" in the hierarchy when control passes from the called to the calling procedure; TSO itself is at the top.

Entering EXIT causes control to go up one level. When EXIT is entered with the QUIT operand, the system attempts to pass control upward to the first procedure encountered that has MAIN or NOFLUSH in effect (see CONTROL Statement). If no such procedure is found, control passes up to TSO, the input stack is flushed of all command procedures, and control passes to the terminal.

```
[label:]        EXIT   [CODE(expression)]
                       [QUIT]
```

label:
specifies a name to which the command procedure can branch. Enter one-to-eight characters, the first alphabetic and the rest alphameric, followed by a colon and at least one blank.

CODE(expression)
specifies a user-defined return code for this exit, with the code specifiable in most simple form as a number or in most complex form as a simple expression (see "Definitions of Command Procedure Terminology"). When CODE is not specified, the system uses the contents of & LASTCC.

QUIT
specifies that control is passed up the nested hierarchy until a procedure is found with the MAIN or NOFLUSH option active or until TSO receives control.

GETFILE Statement

The GETFILE statement allows the user to get a record from an open QSAM file. One record is obtained for one execution of GETFILE. You must know the filename(ddname) by which you allocated and opened (via OPENFILE) the file for this terminal session.

After GETFILE executes, the file variable &filename contains the record obtained.

File variables are scanned only once (no rescans) and only on OPENFILE.

[label:] **GETFILE** **&filename**

---

label:
specifies a name to which the command procedure can branch. Enter one-to-eight characters, the first alphabetic and the rest alphameric, followed by a colon and at least one blank.

&filename
specifies the ddname by which the file was allocated and opened (via OPENFILE).

The GLOBAL statement must precede any statement that uses its variables. The GLOBAL statement defines unique symbolic variables that will be used globally, which in the application means in all lower nested levels of the hierarchy. The first-level command procedure defines global variables; lower-level procedures must include a GLOBAL statement if they intend to refer to the global variables specified in the first level. The number of global variables defined in the first-level procedure is the maximum number that can be referenced by any lower-level procedure.

The global variables are positional, both in the first-level procedure and in all lower-level procedures that reference this same set of variables. This means that the Nth name on any level GLOBAL statement refers to the same variable, even though the symbolic name at each level may be different. Note, however, that the names must still be unique among those at that level.

Since the global variables are symbolic variables, they must have an & prefix except in READ and READDVAL statements, where the & is optional.

[label:] **GLOBAL name1** [name2 . . . . nameN]

---

label:
specifies a name to which the command procedure can branch. Enter one-to-eight characters, the first alphabetic and the rest alphameric, followed by a colon and at least one blank.

name1-nameN
specify valid symbolic variable names for this procedure.

Example

Operation: Specify a set of global variables for three levels of procedures, where some names are unique to their level.

First-level procedure:	GLOBAL	NAME1	NAME2	NAME3	NAME4
Second-level procedure:	GLOBAL	FIRST	SECOND	THIRD	
Third-level procedure:	GLOBAL	PARM1	PARM2	PARM3	PARM4

Note that & NAME3, & THIRD, and & PARM3 would access the same variable.

The GOTO statement causes an unconditional branch within a command procedure. Branching to another command procedure is not supported. When GOTO is specified, control passes to the statement or command that has the label called out as the target.

[label:] **GOTO target**

---

label:
specifies a name to which the command procedure can branch. Enter one-to-eight characters, the first alphabetic and the rest alphameric, followed by a colon and at least one blank.

target
specifies either a label or an expression that reduces to a valid label value after symbolic substitution.

Example

Operation: Illustrate branching within a command procedure.

```
BEGIN:  SET &RET=NEXT
        GOTO LAB1
NEXT:   WRITENR TWO,
        SET &N=2
        GOTO LAB&N
        .
        .
        .
LAB1:   WRITENR ONE,
        GOTO &RET
LAB2:   WRITE THREE
        EXIT    /* ONE,TWO,THREE HAS BEEN WRITTEN
                   TO THE TERMINAL*/
```

IF-THEN-ELSE Statement

The IF-THEN-ELSE sequence defines a condition, tests the truth of that condition, and initiates an action based on the test results.

Note that a continuation character is required if the THEN or ELSE statement extends to the next line. If no continuation character is present and no other text is on the same line, the THEN and ELSE will be treated like null statements.

[label:]	**IF logical-expression THEN** [action] [ELSE [action]]

label:
specifies a name to which the command procedure can branch. Enter one-to-eight characters, the first alphabetic and the rest alphameric, followed by a colon and at least one blank.

logical-expression
is a group of comparative expressions grouped by logical operators (see "Definitions of Command Procedure Terminology"). The minimal entry for logical-expression is a comparative expression.

action
specifies an executable statement, which includes commands, subcommands, and command procedure statements. The THEN action is invoked if the IF condition is satisfied. The ELSE action is invoked if the IF condition is not satisfied and ELSE is specified. If the IF condition is not satisfied and ELSE is not specified, control passes to the next sequential statement.

OPENFILE Statement

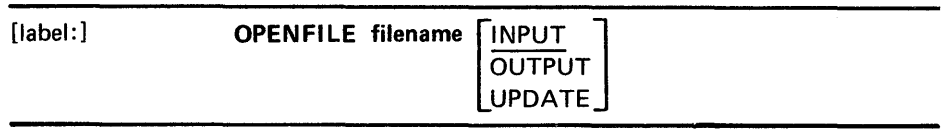
The OPENFILE statement opens a specific file for QSAM I/O. One execution of OPENFILE opens one file. File variables are scanned only once (no rescans) and only on OPENFILE.

Complete your file I/O on a specific file before you change modes from command to subcommand or vice versa. Crossmode file I/O is not supported and will cause miscellaneous abnormal terminations.

Specify NOFLUSH (see the CONTROL statement) for a command procedure that uses file I/O.

If a system action causes you to be flushed because you did not specify NOFLUSH, you will have to log off the system to recover. You will recognize the condition by getting a message similar to "FILE NOT FREED, DATA SET IS OPEN."

For reference information on QSAM I/O, see *OS/VS2 Data Management Services Guide*.



label:
specifies a name to which the command procedure can branch. Enter one-to-eight characters, the first alphabetic and the rest alphameric, followed by a colon and at least one blank.

filename
specifies the ddname of a file that has been previously allocated by the TSO ALLOCATE command or by step allocation. The filename becomes a symbolic variable that will contain either:

- The results of a GETFILE, or
- A record that was set by the user for a PUTFILE.

The filename name does not have to be previously defined.

INPUT
specifies that the filename will open for input. The default is INPUT when neither INPUT, OUTPUT, nor UPDATE is entered.

OUTPUT
specifies that the filename will open for output.

UPDATE
specifies that the filename will open for updating in place; that is, you can replace a previously read record by issuing a PUTFILE statement.

PROC Statement

The PROC statement defines the parameters that can be passed to the command procedure via the value-list parameter of the EXEC command. PROC is optional for a command procedure, but if it is used, it must be the first statement in the command procedure.

Note that a label cannot be entered for a PROC statement.

PROC	**positional-specification**
	[positional-parameters]
	[keyword-parameters [(values)]]

positional-specification
specifies the number of required positional parameters to be passed. Enter 1-5 decimal digits. Enter 0 if none.

positional-parameters
specifies the positional parameters, in sequence, that require initial values in the value list before the command procedure is invoked. Parse will prompt for an initial value if one is not there, except when positional-specification=0 and no prompting is needed because there are no positional parameters.
Positional parameter names are 1-252 characters, the first alphabetic and the rest alphameric. The values must be character strings *without* delimiters.

keyword-parameters(values)
specify the keyword parameters, either with or without values, that do not require initial values in the value list before the command procedure is invoked.
Keyword parameter names are 1-31 characters, the first alphabetic and the rest alphameric. Keywords without values have nothing appended. Keywords with values have the values enclosed in parentheses and appended to their names. A value can be a null entry (keep parentheses), a quoted character string, or an unquoted character string. A quoted character string can include delimiters. These values are defaults and are used when a keyword name is not valid and a value is required.

Note: All symbolic parameters have an initial value at the time the command procedure begins execution. The symbolic parameter value can be changed dynamically by specifying the symbolic parameter name on the READ, SET or READDVAL statements.

PUTFILE Statement

The PUTFILE statement puts a record into an already open QSAM file. One execution of PUTLINE transfer one record. This record must be initialized each time by an assignment statement such as SET unless you want the same record sent more than once. You must know the filename(ddname) by which you allocated and opened (via OPENFILE) the file for this terminal session.

File variables are scanned only once (no rescans) and only on OPENFILE.

[label:] **PUTFILE filename**

---

label:
specifies a name to which the command procedure can branch. Enter one-to-eight characters, the first alphabetic and the rest alphameric, followed by a colon and at least one blank.

filename
specifies the ddname by which the file was allocated and opened (via OPENFILE). The record that is put is the value of the file variable & FILENAME.

Example

Operation: Illustrate typical file I/O.

```
.
.
OPENFILE MYOUTPUT OUTPUT
.
.
SET &MYOUTPUT = TEXT STRING
PUTFILE MYOUTPUT /* TEXT STRING is put to the file */
.
.
```

READ Statement

The READ statement makes terminal user input available to the command procedure as values in symbolic variables. These variables may be named in the READ statement or already named elsewhere in the command procedure. The READ statement is usually preceded by a WRITE to the terminal to identify the expected input.

[label:] **READ** [name1 [name2 . . . nameN]]

---

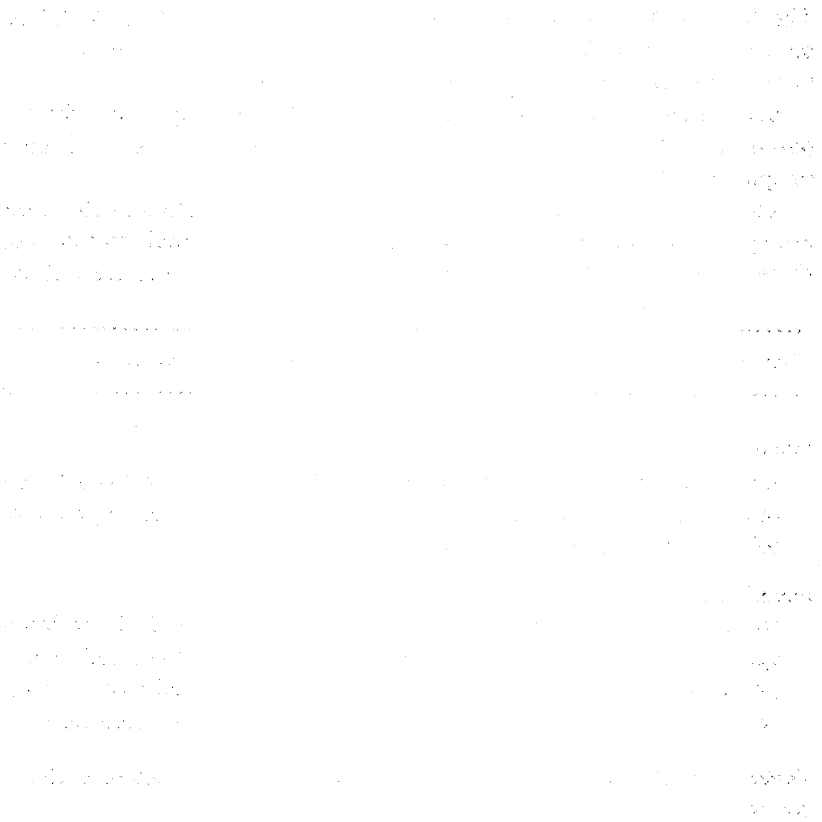
label:
specifies a name to which the command procedure can branch. Enter one-to-eight characters, the first alphabetic and the rest alphameric followed by a colon and at least one blank.

Note: If READ is entered without parameter names, the value of the terminal input line is read into & SYSDVAL.

name1-nameN
specify any syntactically valid parameter names; the & prefix is optional. These symbolic parameters need not be previously defined. The parameters are positional in the sense that recognizable values entered by the command procedure user are set sequentially into the names specified here. Recognizable values are:

- A character string
- A quoted string
- A parenthesized string
- A null value, specified by entering two adjacent commas (,,) or two adjacent quotes (' '). Double quotes (") will not work.

Any or all of the types specified may be entered on one READ statement.



READDVAL Statement

The READDVAL statement causes the current value of & SYSDVAL to be parsed into syntactical words and assigns these words to the symbolic parameters specified on the READDVAL statement.

Syntactical words are defined as character strings, quoted strings, parenthesized strings, or null values indicated by two adjacent commas (,,) or quotes (' ').

The assignment is done sequentially on the parameters in the order they are specified; parameters not assigned a value will default to null values. If there are more words than parameters, the leftover words are not assigned.

[label:] **READDVAL** [name1 [name2 . . . . nameN]]

---

label:
specifies a name to which the command procedure can branch. Enter one-to-eight characters, the first alphabetic and the rest alphameric, followed by a colon and at least one blank.

name1-nameN
specify any syntactically valid parameter names; the & is optional. These symbolic parameters need not have been previously defined. The parameters are positional in the sense that syntactical words from & SYSDVAL are set sequentially into the names specified here.

Note: If READDVAL is entered without symbolic parameters, the statement is ignored.

RETURN Statement

The RETURN statement specifically returns control from an error range or attention range to the statement following the one that ended in error or the one that was interrupted by an attention.

RETURN is valid only when issued from an activated error action range or an activated attention action range from this command procedure. If neither of these conditions exists, the RETURN is treated as a no-operation.

[label:] **RETURN**

---

label:
specifies a name to which the command procedure can branch. Enter one-to-eight characters, the first alphabetic and the rest alphameric, followed by a colon and one or more blanks.

SET Statement

The SET statement assigns a specified value to a specified symbolic variable name. One value is assigned to one variable for one execution of SET. The variable need not have been predefined elsewhere.

The variable to be set cannot be a built-in function.

[label:] **SET symbolic-variable-name** { **=** / **EQ** } **expression**

---

label:
specifies a name to which the command procedure can branch. Enter one-to-eight characters, the first alphabetic and the rest alphameric, followed by a colon and at least one blank.

symbolic-variable-name
specifies the syntactically valid symbolic variable or allowable control variable that is to be set.

EQ or =
specifies the comparison operator EQUAL.

expression
specifies a simple expression as defined in "Definitions of Command Procedure Terminology."

TERMIN Statement

The TERMIN statement passes control from the command procedure currently executing to the terminal user. TERMIN also defines the character strings that a user can enter to return control to the command procedure. A null value can be specified as a character string that the user can enter. TERMIN is usually preceded by a WRITE statement that identifies the expected response to the terminal user.

Control returns to the command procedure at the statement after TERMIN. When control returns, & SYSDLM and & SYSDVAL have been set.

[label:] **TERMIN** [string1 / ,] [string2 . . . . stringN]

---

label:
specifies a name to which the command procedure can branch. Enter one-to-eight characters, the first alphabetic and the rest alphameric, followed by a colon and at least one blank.

string1-stringN
specify character strings that the terminal user can enter to return control to the command processor. The & SYSDLM control variable contains the number of the string which was entered (1 for string1, 2 for string2, etc.) and & SYSDVAL contains the balance of the entered line.

,(comma)
can be used only in the first string position and specifies that the terminal user can enter a null line to return control to the command procedure.

WRITE and WRITENR Statements

The WRITE and WRITENR statements send text to the terminal user from the command procedure. Thus text can be used for messages, information, prompting, or whatever the writer of the command procedure wishes.

[label:] **WRITE**[NR] **text**

---

label:
specifies a name to which the command procedure can branch. Enter one-to-eight characters, the first alphabetic and the rest alphameric, followed by a colon and at least one blank.

WRITE
statement specifies that the cursor moves to a new line after the text is displayed.

WRITENR
statement specifies that the cursor does not move to a new line after the text is displayed.

text
specifies what is to be sent to the terminal. You can enter any character string, including symbolic variables. Data enclosed within /* and */ delimiters is also sent to the terminal even though it may appear as a comment.

Example

Operation: Illustrates WRITE and WRITENR usage.

```
.
.
WRITENR ONE
WRITENR TWO/
WRITENR THREE
WRITE FOUR
WRITE FIVE
.
.
```

The display at the terminal will be:

```
ONETWO/THREEFOUR
FIVE
```

Appendix A: Foreground-Initiated Background Commands

You may use the foreground-initiated background (FIB) commands to submit or control jobs for execution in a batch environment.

Using Foreground-Initiated Background (FIB) Commands

Use CANCEL, OUTPUT, STATUS and SUBMIT commands primarily to control the submission and processing of jobs in a batch environment. Also, the OUTPUT command may be used to control foreground-created output.

Processing Batch Jobs

You can submit batch jobs for processing if your installation authorizes you to do so. This authorization is recorded in the system with your user attributes. If you have this authorization, the system lets you use the four commands (SUBMIT, STATUS, CANCEL and OUTPUT) that control the processing of batch jobs. You can use those commands to submit a batch job, to display the status of a batch job, to cancel a batch job, and to control the output of a batch job.

Submitting Batch Jobs

Before you submit a batch job with the SUBMIT command you can use the EDIT command to create a data set (or a member of a partitioned data set) that contains the job or jobs you want to submit. Each job consists of job control language (JCL) statements and of program instructions and data.

The first JCL statement in the data set is usually a JOB statement. The jobname in the JOB statement can be up to eight characters in length and should consist of your user identification followed by one or more letters or numbers, for example, SMITH23 or JONESXYZ.

If the jobname does not begin with your user identification, you can submit it with the SUBMIT command and request its status with the STATUS command, but you cannot refer to it with the CANCEL or OUTPUT command unless the IBM-supplied installation exit is replaced.

If the jobname consists of only your user identification, the system will prompt you for one or more characters to complete the jobname. This allows you to change jobnames without re-editing the data. For example, you may submit the same job several times, and supply a different character for the job name each time you are prompted.

If the first non-job entry subsystem statement of your data set is not a JOB statement, the system generates the following JOB statement when you submit it with the SUBMIT command.

```
//userid JOB accounting info,
//     userid, ** JOB STATEMENT GENERATED BY SUBMIT **
//     NOTIFY=userid,
//     MSGLEVEL=(1,1)
```

You will be prompted for a character to complete the jobname. The job accounting information is the information specified by the user at LOGON.

When you enter the SUBMIT command, you must give the name of a data set (or data sets) containing the batch job (or jobs). You can also specify the NONOTIFY operand to specify that you do not want to be notified when a batch job with a generated JOB statement terminates.

Figure 19 shows how to create and submit a batch job. The data set type on the EDIT command should be CNTL for better system performance. The SUBMIT command will perform best if the fully-qualified data set name is entered in quotes. Submitted data sets must have a logical record length of 80 bytes, a record format of fixed-blocked (FB), and must not contain lowercase characters.

You may include more than one job in one data set. You can omit the JOB statement for the first job, but all jobs after the first must have their own JOB statement. Although you submit all jobs in the data set with one SUBMIT command, you can subsequently refer to each job with separate STATUS, CANCEL, and OUTPUT commands.

When you submit more than one job with a single command, and TSO finds an error while processing the first job, the second job is not processed. An error that occurs in the second job does not affect the first. Any jobs processed prior to the error are submitted for execution; jobs that were not processed because of the error should be resubmitted after the error is corrected.

```
    READY
    Edit    backpgm new     cntl
    INPUT
0010//smith3        job     7924,smith,msglevel=(1,1),
0020//              notify=smith3
0030//step1         exec    pl1lfc,parm.pl1l='nodeck,list'
0040//pl1l.sysin    dd      *
0050    .   source statement
0060    .
0070    .
0080/*
0090//step2         exec    pl1lfclg
0100//pl1l.sysin    dd      *
0110    .   source statements
0120    .
0130    .
0140/*
0150//go.sysin      dd      *
0160    .
0170    .
0180    .
0190    .   input data
0200    .
0210    .
0220/*
    (null line)
    EDIT
    end save
    READY
    submit  backpgm
    ENTER JOBNAME CHARACTERS+ -
    a
    JOB SMITH3A(JOB00071) SUBMITTED
    READY
```

Figure 19. Submitting a Program as a Batch Job

The user would get a job-ended message with a time stamp at the terminal because the NOTIFY keyword is specified on the JOB card.

A submitted data set need not contain an entire job. A JCL data set and a source data set could be used if both were the proper type of data set, as follows:

```
submit (jclds1 sourceds jclds2 sourceds)
```

If each JCL data set contained a job card, then two jobs would be submitted above. JCLDS1 could contain the JCL needed to print the source data set following in the input stream and JCLDS2 could contain the JCL needed to assemble the same data set.

Displaying the Status of Jobs

Any time after you submit a background job you can use the STATUS command to have its status displayed. The display will tell you whether the job is awaiting execution, is currently executing, or has executed but is still on the output queue. The display will also indicate whether a job is in hold status. For example, if you want to display the status of SMITH3A, enter:

```
READY
status smith3a
```

If you have submitted two jobs with jobname SMITH3A, but just want the status of the job submitted in Figure 19, you should enter the jobid with the jobname, as follows:

```
READY
status smith3a(job71)
```

If you want to know the status of all the jobs with jobnames consisting of your user identification plus one character, enter the STATUS command without operands:

```
READY
status
```

You may also check the status of data sets held from previous foreground sessions by using the STATUS command.

Cancelling Batch Jobs

The CANCEL command cancels execution of a batch job. For example, if you want to cancel job JONESAB, and cancel its output if it has already executed, enter:

```
READY
cancel jonesab,p
```

After you enter the CANCEL command, the system will send you a READY message and will notify the operator that the job has been cancelled.

Controlling the Output of Batch or Foreground Jobs

The OUTPUT command may be used to manipulate all held output, regardless of whether the output is produced during the current LOGON session, a previous LOGON session, or by a batch job submitted from any source. This output must be held for terminal access in one of two ways:

- Explicitly via HOLD=YES on a DD statement or via the ALLOCATE or FREE command, for example:

```
//SMITH6    JOB     MSGLEVEL=1,MSGCLASS=C,NOTIFY=SMITH
//          EXEC        PGM=IEBDG
//SYSPRINT  DD      SYSOUT=M,HOLD=YES
//
//  remainder of JCL statements
//
```

or

- Implicitly by specifying an installation-defined reserved class for SYSOUT and MSGCLASS. It is not necessary to have them reserved in the same class. For example:

```
//SMITH6    JOB     MSGLEVEL=1,MSGCLASS=R,NOTIFY=SMITH
//          EXEC    PGM=IEBDG
//SYSPRINT  DD      SYSOUT=S
//
//  remainder of JCL statements
//
```

The OUTPUT command can:

- Direct the JCL statements, system messages (MSGCLASS), and system output data sets (SYSOUT) produced by a job to your terminal.
- Direct the MSGCLASS and SYSOUT output from a job to a specific data set.
- Change an output class used in a job.
- Route the MSGCLASS and SYSOUT output from a job to a remote station.
- Release the output of a job for printing.
- Delete the output data sets (SYSOUT) or the system messages (MSGCLASS) for jobs.

If you have NOTIFY=userid on the job cards that were submitted, a message is written to your terminal or placed in the broadcast data set when the background job terminates. Provided you have held the output, you can then use the OUTPUT command to control the held output produced by the job.

For example, assume that job GREEN67 produces held output in classes A, B, D, M, G, and 6. If you want the output in classes G and M displayed at the terminal, enter:

```
READY
output green67 class(g m) print(*)
```

If you want the output of class B to be listed in the GREEN.KEEP.OUTLIST data set, enter:

```
READY
output green67 class(b) print(keep)
```

If you want to change the output in class A to class C, enter:

```
READY
output green67 class(a) newclass(c)
```

If you want to delete the output from class D, enter:

```
READY
output green67 class(d) delete
```

If you want to release the output of class 6, and have it printed in the background by output services, enter:

```
READY
output green67 class(6) nohold
```

You can enter the PAUSE operand in the OUTPUT command to make the system stop after each data set is displayed on your terminal or on the data set you indicate with the PRINT operand. When the system pauses it sends you the message OUTPUT. You then have the option of pressing the ENTER key to continue processing or entering the CONTINUE, SAVE, END or HELP subcommand.

The CONTINUE subcommand allows you to continue processing your output after an interruption occurs. An interruption occurs when:

- The printing of a data set is completed and you used the PAUSE operand in the OUTPUT command.
- You press the attention interruption key.

Note: An attention interruption can cause unpredictable results in the print processing. When the attention interruption key is hit, the data set may be checkpointed 10 to 20 records back.

To retrieve data created during previous LOGON sessions, issue STATUS userid. STATUS will return a jobid and status for each LOGON session as a job on the output queue. It will also return jobid and status for the current LOGON session as a job in execution.

When you enter the CONTINUE subcommand, the system will resume the display with the next data set to be processed. In the following example you request that the held data sets in output classes B and C be displayed at your terminal. The system pauses after displaying the data set in B. You enter the CONTINUE subcommand to resume processing with data set in C.

```
READY
output jones2 class(b c) print(*) pause
.
.
.   output class B
.
.
OUTPUT
continue
.
.
.   output class C
.
.
```

If the interruption was not caused by a pause, you may prefer to resume displaying at the beginning of the data set being processed. To resume displaying at the beginning, enter:

```
OUTPUT
continue begin
```

If you prefer to resume displaying approximately 10 lines before the interruption occurred, enter:

```
OUTPUT
continue here
```

The CONTINUE subcommand also lets you respecify the PAUSE operand of the OUTPUT command. If you entered PAUSE in the OUTPUT command, you can enter NOPAUSE in the CONTINUE subcommand, for example:

```
READY
output smithc class(d) print(data) pause
.
.
.
OUTPUT
continue begin nopause
```

If you did not specify PAUSE in the OUTPUT command, you can do so in the CONTINUE subcommand. This causes the system to pause at the end of each data set processed subsequently.

The SAVE subcommand allows you to place the data set listed before the pause into another data set. This allows you to retrieve the data set later. In the following example, if your LOGON identifier is Brown, you request that held data sets in output classes E and F be listed at your terminal. After listing the data set in E you request that it be saved in the BROWN.OUTDATA.OUTLIST data set. You continue processing the next data set after saving the data set in class E.

Note: If you want to display output at a terminal when submitting one or more jobs, the name you specify must begin with your userid and optionally end with one or more alphameric characters (if the IBM-supplied installation exit is used).

```
READY
output brownb class(e f) print(*) pause
.
.
.
OUTPUT
save    outdata
OUTPUT
continue
.
.
.
```

The END subcommand is used to terminate the OUTPUT command. For example:

```
READY
output dept30a class(a) print(*) pause
.
.
.
OUTPUT
end
READY
```

CANCEL Command

Use the CANCEL command to halt processing of batch jobs that you have submitted from your terminal. A READY message will be displayed at your terminal if the job has been canceled successfully. A message will also be displayed at the system operator's console when a job is canceled.

Installation management must authorize the use of CANCEL. This command is generally used in conjunction with the SUBMIT, STATUS, and OUTPUT commands.

CANCEL (jobname[(jobid)]-list)
[NOPURGE / PURGE]

(jobname[(jobid)]-list)
specifies the names of the jobs that you want to cancel. The jobnames must consist of your user identification plus one or more alphameric characters up to a maximum of eight characters unless the IBM-supplied exit has been replaced by your installation.
Also, you cannot cancel a TSO user or a started task that is not on an output queue. The optional jobid subfield may consist of one to eight alphameric characters (the first character must be alphabetic or national). The jobid is a unique job identifier assigned by the job entry subsystem at the time the job was submitted to the batch system. The jobid is needed if you have submitted two jobs with the same name.

Notes:

- 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.
- Jobs controlled by the subsystems are considered started tasks and cannot be cancelled via the CANCEL command.

PURGE
specifies that the job and its output (on the output queue) are to be purged from the system.

NOPURGE
specifies that jobs are to be canceled if they are in execution; output generated by the jobs will remain available. If the jobs have executed, the output still remains available.

Example 1

Operation: Cancel a batch job.

Known:
The name of the job: JE024A1

```
cancel je024a1
```

Example 2

Operation: Cancel several batch jobs.

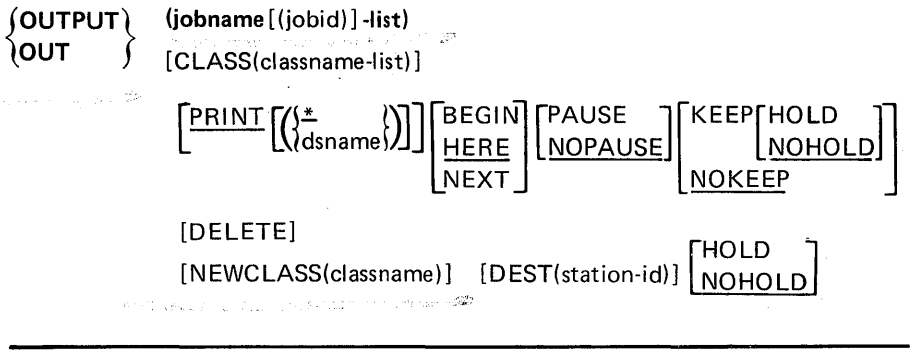
Known:
The names of the jobs: D58BOBTA D58BOBTB(J51) D58BOBTC

```
cancel (d58bobta d58bobtb(j51) d58bobtc)
```

OUTPUT Command

Use the OUTPUT command to:

- Direct the output from a job to your terminal. The output includes the job's job control language statements (JCL), system messages (MSGCLASS), and system output (SYSOUT) data sets.
- Direct the output from a job to a specific data set.
- Delete the output for jobs.
- Change the output class(es) for a job.
- Route the output for a job to a remote work station.
- Release the output for a job for printing by the subsystem.



(jobname[(jobid)]-list)
specifies one or more names of batch or foreground jobs. The jobname for foreground session is userid. Each jobname must begin with your user identification and, optionally, can include one or more additional characters unless the IBM-supplied installation exit that scans and checks the jobname and user identification is replaced by a user-written routine. The system will process the held output from the jobs identified by the job-name-list. You should include the optional jobid for uniqueness to avoid duplicate jobnames.

CLASS(classname-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 a class, all held output for the jobs will be available. A class name is a single character or digit (A-Z or 0-9).

PRINT(dsname or *)
specifies the name of the data set to which the output is to be directed. If unqualified, the data-set-name will have the user prefix added and the qualifier OUTLIST appended to it. 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 PRINT, DELETE, NEWCLASS, DEST, and HOLD/NOHOLD.
If the PRINT data set is not pre-allocated, RECFM defaults to FBS, LRECL defaults to 132, and the BLKSIZE defaults to 3036.

BEGIN
indicates that output operations for a data set are to start from the beginning of the data set whether it has been checkpointed or not.

HERE
indicates that output operations for a data set that has been checkpointed are to be resumed at the approximate point of interruption. If the data set is not checkpointed, it will be processed from the beginning. HERE is the default value if you omit HERE, BEGIN, and NEXT.

NEXT
indicates that output operations for a data set that has been previously checkpointed are to be skipped. Processing resumes at the beginning of the uncheckpointed data sets. *Caution:* The checkpointed data sets that are skipped will be deleted unless the KEEP operand is specified.

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. (Pressing the ENTER key 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. This is the default if neither PAUSE nor NOPAUSE is specified.

KEEP
specifies that the SYSOUT data set will remain enqueued after printing (see also HOLD and NOHOLD).

NOKEEP
specifies that the SYSOUT data set be deleted after it is printed. NOKEEP is the default if neither KEEP nor NOKEEP is specified.

HOLD
specifies that the kept SYSOUT data set be held for later access from the terminal.

NOHOLD
specifies that the kept SYSOUT data set be released for printing by the subsystem. This is the default for KEEP if neither HOLD nor NOHOLD is specified.

DELETE
specifies that the classes of output specified with the CLASS operand are to be deleted.

NEWCLASS(classname)
is used to change one or more SYSOUT classes to the class specified by the "classname" subfield.

DEST(station id)
routes SYSOUT classes to a remote work station specified by the "station id" subfield.

Note: Unless 5740-XT60 is installed, the DEST operand is 7 characters; otherwise it is 8 characters.

Considerations: The OUTPUT command applies to all 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 an installation-written exit routine. The SUBMIT, STATUS, and CANCEL commands 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 either on the JOB card or on the SUBMIT command when you submit a job for batch processing. The system will notify you when the job terminates, giving you an opportunity to use the OUTPUT command. MSGCLASS and SYSOUT data sets should be assigned to reserved classes or explicitly held in order to be available at the terminal.

Output Sequence: Output will be produced according to the sequence of the jobs that are specified, then by the sequence of classes that are specified 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=3330,
//              VOL=SER=11112,LABEL=( ,SUL),
//              DIPS=(OLD,KEEP),
//              DCB=(RECFM=U,BLKSIZE=3036)
//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=3330,VOL=SER=333000,
//              DISP=OLD
//SYSIN         DD      *
            SCRATCH VTOC,VOL=3330=333000
/*
```

To retrieve the output, you enter:

```
output  (jwsd581 jwsd582) class (x y z)
```

Your output will be displayed in the following order:

1. Output of job JWSD581
   a. class X (JCL and messages)
   b. class Y (SYSPRINT data)
   c. class Z (SYSUT2 data)
2. Output of job JWSD582
   a. class X (JCL and messages)
   b. class Y (SYSPRINT data)
   c. message (No CLASS Z OUTPUT FOR JOB JWSD582)

If no classes are specified, the jobs will be processed as entered. Class sequence is not predictable.

Subcommands: Subcommands for the OUTPUT command are: CONTINUE, END, HELP, and SAVE. When output has been interrupted, you can use the CONTINUE subcommand to resume output operations.

Interruptions causing subcommand mode occur when:

- Processing of a sysout data set completes and the PAUSE operand was specified with the OUTPUT command.
- You press the attention key.

Note: Pressing the attention key purges the input/output buffers for the terminal. Data and system messages in the buffers at this time may be lost.

Although the OUTPUT command attempts to back up 10 records to recover the lost information, results are unpredictable due to record length and buffer size. The user may see records repeated or he may notice records missing if he attempts to resume processing of a data set at the point of interruption (using the HERE operand of CONTINUE, or in the next session using HERE on the command).

You can use the SAVE subcommand to copy a SYSOUT data set to another 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 kept for later retrieval at the terminal.

Checkpointed Data Set: A data set is checkpointed if it is interrupted during printing and never processed to end of data during a terminal session.

Interruptions which cause a data set to be checkpointed occur when:

- Processing terminates in the middle of printing a data set because of an error or ABEND condition.
- The attention key is pressed during the printing of a data set and the CONTINUE NEXT subcommand is entered. The KEEP operand must be present or the data set will be deleted.
- The attention key is pressed during the printing of a data set and the END subcommand is entered.

Example 1

Operation: Direct the held output from a job to your terminal. Skip any checkpointed data sets.

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 that have never been interrupted. You want the system to pause after processing each output data set.

```
output smith2 class(x) print(*) next pause
```

Example 2

Operation: Direct the held output from two jobs to a data set so that it can be saved and processed at a later date.

Known:
The name of the jobs: JANA JANB
The name for the output data set: JAN.AUGPP.OUTLIST

```
output (jana,janb) class(r,s,t) print(augpp)
```

Example 3

Operation: Change an output class.

Known:
The name of the job: KEAN1
The existing output class: SYSOUT=S
The new output class: T

```
output kean1 class(s) newclass(t)
```

Example 4

Operation: Delete the held output instead of changing the class (see Example 3).

```
out kean1 class(s) delete
```

Example 5

Operation: Retrieve SYSOUT data from your session at your terminal.

Known:
The TSO userid: SMITH
A JES held SYSOUT class: X
The filename of the SYSOUT data set: SYSUT2

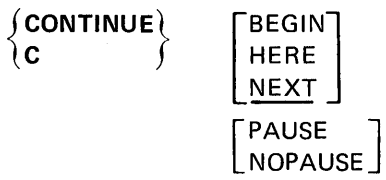
```
free file(sysut2) sysout(x)
status smith
SMITH(TSU0001) EXECUTING
output smith(tsu0001)
```

CONTINUE Subcommand of OUTPUT

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.
- You press the attention key.



BEGIN
indicates that output operations are to be resumed from the beginning of the data set being processed at the time of interruption.

NEXT
halts all processing of the current data set and specifies that output operations are to be resumed with the next data set.
The next data set is determined by the BEGIN, HERE, or NEXT operand on the OUTPUT command. If BEGIN was specified on the command, processing will start at the beginning of the next data set. If HERE was specified, processing will start at the checkpoint of the next data set, or at its beginning if no checkpoint exists. If NEXT was specified, processing will start at the beginning of the next uncheckpointed data set. NEXT is the default value if BEGIN, HERE, and NEXT are omitted.

Note: The data set that was interrupted and any that are skipped will be deleted unless KEEP was specified on the command.

HERE
indicates that output operations are to be resumed at a point of interruption. If the attention key was pressed, processing resumes at the approximate point of interruption in the current data set. If end of data was reached and PAUSE was specified, processing resumes at the beginning of the next data set (even if it was checkpointed and HERE was specified on the command).

PAUSE
indicates that output operations are to pause after each data set is processed to allow you to enter a SAVE subcommand. (Pressing the ENTER key 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 condition for output.

Example 1

Operation: Continue output operation with the next SYSOUT data set.

```
continue
```

Example 2

Operation: Start output operations over again with the current data set being processed.

```
continue begin
```

END Subcommand of OUTPUT

Use the END subcommand to terminate the operation of the OUTPUT command.

END

HELP Subcommand of OUTPUT

Use the HELP subcommand to obtain the syntax and function of the OUTPUT subcommands. Refer to the HELP command for a description of the syntax and function of the HELP subcommand.

SAVE Subcommand of OUTPUT

Use the SAVE subcommand to copy the SYSOUT data set from the spool data set to the named data set. This data set can be any data set that would be valid if used with the PRINT operand. There is no restriction against saving JCL. To use SAVE, you should have specified the PAUSE keyword on the OUTPUT command. SAVE will not save the entire SYSOUT output of the job, only the data set currently being processed.

{SAVE / S} **data-set-name**

---

data-set-name
specifies the new data set name to which the SYSOUT data set is to be copied.

Example 1

Operation: Save an output data set.

Known:
The name of the data set: ADT023.NEWOUT.OUTLIST

```
save newout
```

Example 2

Operation: Save an output data set.

Known:
The name of the data set: BXZ037A.OLDPART.OUTLIST
The data set member name: MEM5
The data set password: ZIP

```
save    oldpart(mem5)/zip
```

STATUS Command

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 but is still on an output queue. It will also indicate whether the job is in hold status.

This command may be used only by personnel who have been given the authority to do so by the installation management.

{STATUS / ST} [(jobname[(jobid)]-list)]

(jobname[(jobid)]-list)
specifies the names of the conventional batch jobs for which you want to know the status. If two or more jobs have the same jobname, the system will display the status of all the jobs encountered and supply jobids for identification. 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 consist of your userid and one identifying character (alphameric or national).
The optional jobid subfield may consist of one to eight alphameric characters (the first character must be alphabetic or national). The jobid is a unique job identifier assigned by the job entry subsystem at the time the job was submitted to the batch system.

Note: When you specify a list of job names, you must separate the jobnames with standard delimiters.

SUBMIT Command

Use the SUBMIT command to submit one or more batch jobs for conventional processing. Each job submitted must reside in either a sequential data set, a direct-access data set, or in a member of a partitioned data set. Submitted data sets must be fixed blocked, 80 byte records. Using the EDIT command to create a CNTL data set will provide the correct format.

Any of these data sets can contain part of a job, one job, or more than one job that can be executed via a single entry of SUBMIT. Each job must comprise an input job stream (JCL plus data). Do not submit data sets with descriptive qualifiers TEXT or PLI if the characters in these data sets are lower case.

Job cards are optional. The generated jobname will be your userid plus an identifying character. SUBMIT will prompt you for the character and will insert the job accounting information from the user's LOGON command on any generated job card. The system or installation default MSGCLASS and CLASS are used for submitted jobs unless MSGCLASS and CLASS are specified on the job card(s) being submitted. See the first section in Appendix A for an example of a generated JOB card.

{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 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 a JOB statement has not been provided. 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. NOTIFY is the default value if a JOB statement is generated.
When you supply your own JOB statement, use the NOTIFY=userid keyword on the JOB statement if you wish to be notified when the job terminates. SUBMIT ignores the NOTIFY keyword unless it is generating a JOB statement.

NONOTIFY
specifies that a termination message will not be issued or placed in the broadcast data set. The NONOTIFY keyword is only recognized when a JOB statement has not been provided with the job that you are processing.

Notes:

- If any of the above types of data sets containing two or more jobs is submitted for processing, certain conditions apply.
  The SUBMIT processor will build a job card for the first job in the first data set, if none was supplied, but will not build job cards for any other jobs in the data set(s).
  If the SUBMIT processor determines that the first job contains an error, none of the jobs is submitted. 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.
- Any job card you supply should have a job name consisting of your userid and a single identifying character. If the jobname is not in this format, you will not be able to refer to it with the CANCEL command. You will be required to specify the jobname in the STATUS command if the IBM-supplied exit has not been replaced by your installation and your job name is not your userid plus a single identifying character.
- If you wish to provide a job card but you also want to be prompted for a unique jobname character, put your userid in the jobname field and follow it with blanks so that there is room for SUBMIT to insert the prompted-for character. This allows you to change jobnames without re-editing the JCL data set.
- Once SUBMIT has successfully submitted a job for conventional batch processing, it will issue a 'jobname(jobid) submitted' message. The jobid is a unique job identifier assigned by the job entry subsystem.
- This command may be used only by personnel who have been given the authority to do so by the installation management.
- If SUBMIT is to generate a JOB statement preceding one or more job entry subsystem control cards, make the first card of your data set a comment card. If this is not done, SUBMIT will generate the JOB statement following any job entry subsystem control cards.

Note: Data sets that are dynamically allocated by the SUBMIT command processor are not automatically freed when the command processor terminates. You must explicitly free dynamically allocated data sets.

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
```

```
submit (stress, strain)
```

Example 2

Operation: Data sets may be concatenated and submitted as a single job.

Known:

```
JCL.CNTL(ASMFCLG): contains JCL for the job.
MYDATA.DATA: contains the input data.
```

```
submit  (jcl(asmfclg) mydata)
```

This will cause a single background job to be submitted and will simultaneously concatenate a generated job card (if required), job control language, and the data. Each data set will not be submitted as a separate job.

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 *OS/TSO Assembler Prompter User's Guide*, SC26-3740, for detailed information on this command.

Use the ASM command to process assembler language data sets and produce object modules. The prompter requests required information and enables you to correct your errors at the terminal.

COBOL Command

The COBOL command is provided as part of the optional COBOL Prompter program product, which is available for a license fee. See *IBM OS (TSO) COBOL Prompter Terminal User's Guide and Reference*, SC28-6433, for detailed information on this command.

Use the COBOL command to compile American National Standard (ANS) COBOL programs. This command reads and interprets parameters for the OS Full American National Standard COBOL Version 3 or Version 4 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.

COBOL also allows specification of the TEST operand to compile programs suitable for testing with the COBOL Interactive Debug program product (see TESTCOB command).

CONVERT Command

The CONVERT command is provided as part of the Code and Go FORTRAN program product, which is available for a license fee. See *IBM System/360 OS (TSO) Code and Go FORTRAN Processor Terminal User's Guide*, SC28-6842, for detailed information on this command.

The CONVERT command converts 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 shown in Figure 20.

FROM	TO
Free-form statements suitable for the Code and Go FORTRAN compiler	Fixed-form statements suitable for the FORTRAN compilers.
Fixed-form statements suitable for the FORTRAN (G1) compiler or the Code and Go FORTRAN compiler	Free-form statements suitable for the Code and Go FORTRAN compiler.

Figure 20. Language Conversions Using the CONVERT Command

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 *OS/MVT and OS/VS2 TSO Data Utilities: COPY, FORMAT, LIST, MERGE User's Guide and Reference*, GC28-6765, for detailed information on this command.

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. See *OS/MVT and OS/VS2 TSO Data Utilities: COPY, FORMAT, LIST, MERGE User's Guide and Reference*, SC28-6765, for detailed information on this subcommand.

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. See *OS/MVT and OS/VS2 TSO Data Utilities: COPY, FORMAT, LIST, MERGE User's Guide and Reference*, SC28-6765, for detailed information on this subcommand.

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 *OS/MVT and OS/VS2 TSO Data Utilities: COPY, FORMAT, LIST, MERGE User's Guide and Reference*, SC28-6765, for detailed information on this command.

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 identation.
- 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 TSO FORTRAN Prompter program product, which is available for a license fee. See *IBM System/360 OS (TSO) Terminal User's Supplement for FORTRAN IV (G1) Processor and TSO FORTRAN Prompter*, SC28-6855, for detailed information on this command.

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.

FORT also allows specification of the TEST operand to compile programs suitable for testing with the FORTRAN Interactive Debug program product (see the TESTFORT command).

GOFORT Command

The GOFORT command is provided as part of the optional TSO Code and Go FORTRAN processor. See *IBM System/360 OS (TSO) Code and Go FORTRAN Processor Terminal User's Guide*, SC28-6842, for detailed information on this command. It may be used to compile, load and execute a source program that has previously been saved. The GOFORT command permits the execution of programs initially coded using the BCD character set; neither the RUN command nor the RUN subcommand of EDIT provides this capability.

GOFORT also allows specification of the TEST operand to compile programs suitable for testing with the FORTRAN Interactive Debug program product (see the TESTFORT command).

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 *OS/MVT and OS/VS2 TSO Data Utilities: COPY, FORMAT, LIST, MERGE User's Guide and Reference*, SC28-6765, for detailed information on this command.

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 *OS/MVT and OS/VS2 TSO Data Utilities: COPY, FORMAT, LIST, MERGE User's Guide and Reference*, SC28-6765, for detailed information on this command.

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.

PLI Command

The PLI command is provided as part of the optional PL/I Optimizing compiler program product, which is available for a license fee. See *OS PL/I Optimizing Compiler: TSO User's Guide*, SC33-0029, for detailed information on this command. The program product includes the PL/I Prompter.

Use the PLI command to invoke the PL/I Optimizing compiler. The prompter will allocate required data sets and prompt you for any information that you have omitted or entered incorrectly, then it will pass control to the compiler.

PLIC Command

The PLIC command is provided as part of the optional PL/I Checkout compiler program product, which is available for a license fee. See *OS PL/I Checkout Compiler: TSO User's Guide*, SC33-0033, for detailed information on this command. The program product includes the PL/I Prompter.

Use the PLIC command to invoke the PL/I Checkout compiler. The prompter will allocate required data sets and prompt you for any information that you have omitted or entered incorrectly, then it will pass control to the compiler.

Subcommands of the PLIC command are provided to aid checking-out of the PL/I program. These allow the programmer to intervene during execution of the program and temporarily modify it.

TESTCOB Command

The TESTCOB command is provided as part of the optional COBOL Interactive Debug program product, which is available for a license fee. See *IBM OS COBOL Interactive Debug Terminal User's Guide and Reference*, SC28-6465, for detailed information on this command. Used with Full American National Standard COBOL Version 4, Compiler or the OS/VS COBOL Compiler, COBOL Interactive Debug enables the COBOL programmer to monitor and control the execution of his COBOL program from a terminal. It greatly simplifies the debugging of COBOL object programs by providing facilities that make errors readily apparent and easily correctable.

TESTFORT Command

The TESTFORT command is provided as part of the optional FORTRAN Interactive Debug program product, which is available for a license fee. See *IBM FORTRAN Interactive Debug for OS (TSO) and VM/370 (CMS) Terminal User's Guide*, SC28-6885, for detailed information on this command. Used in conjunction with Code and Go FORTRAN or FORTRAN IV(G1), FORTRAN Interactive Debug provides comprehensive capabilities for program monitoring and checkout.

Appendix C: Access Method Services Commands

Access Method Services is a multifunction service program that primarily establishes and maintains Virtual Storage Access Method (VSAM) data sets (see also Figure 4). The following Access Method Services commands provide the service functions applicable to VSAM data sets and are used in the same way as TSO commands at the terminal:

ALTE changes attributes in catalog entries.

BLDINDEX(BIX) builds alternate indexes.

DEFINE (DEF) creates catalog entries for data sets. Subcommands are:

```
ALIAS
ALTERNATEINDEX( AIX )
CLUSTER( CL )
GENERATIONDATAGROUP( GDG )
NONVSAM( NVSAM )
PAGESPACE( PGSPC )
PATH
SPACE( SPC )
USERCATALOG( UCAT )
```

DELETE (DEL) deletes catalog entries.

EXPORT (EXP) copies a data set for backup.

EXPORTRA (XPRA) makes entries and data from a recoverable catalog portable.

IMPORT (IMP) reads a backup copy of a data set.

IMPORTRA (MPRA) reestablishes entries and data made portable by EXPORTRA.

LISTCAT (LISTC) lists catalog entries.

LISTCRA (LISTR) lists catalog entries in the the catalog recovery area (CRA).

PRINT prints VSAM data sets.

REPRO copies data sets and converts sequential and indexed-sequential data sets to VSAM format.

RESETCAT (RCAT) synchronizes a damaged catalog with specified catalog recovery areas.

VERIFY (VFY) causes a catalog to correctly record the end of a data set after a data set closing error may have caused the end to be recorded incorrectly.

CNVTCAT converts the contents of an OS catalog or control volume into entries in an MVS or Release 3 catalog.

For additional information about the syntax and function of these commands, refer to *OS/VS2 Access Method Services.*

GC28-0646-4

OS/VS2 TSO Command Language Reference (S370-39) Printed in U.S.A. GC28-0646-4

OS/VS2 TSO Command Language Reference

GC28-0646-4

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate.

IBM shall have the nonexclusive right, in its discretion, to use and distribute all submitted information, in any form, for any and all purposes, without obligation of any kind to the submitter. Your interest is appreciated.

Note: ***Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.***

Possible topics for comments are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If comments apply to a Selectable Unit, please provide the name of the Selectable Unit _____.

If you wish a reply, give your name and mailing address:

_____

_____

_____

Note: Staples can cause problems with automated mail sorting equipment. Please use pressure sensitive or other gummed tape to seal this form.

Cut or Fold Along Line

Please circle the description that most closely describes your occupation.

Customer	(Q) Install Mgr.	(U) System Consult.	(X) System Analyst	(Y) System Prog.	(Z) Applica. Prog.	(F) System Oper.	(I) I/O Oper.	(L) Term. Oper.

(O) Other

IBM	(S) System Eng.	(P) Prog. Sys. Rep.	(A) System Analyst	(B) System Prog.	(C) Applica. Prog.	(D) Dev. Prog.	(R) Comp. Prog.	(G) System Oper.	(J) I/O Oper.	(E) Ed. Dev. Rep.	(N) Cust. Eng.	(T) Tech. Staff Rep.

Number of latest Newsletter associated with this publication: _____

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.)

GC28-0646-4

Reader's Comment Form

Cut or Fold Along Line

Fold and tape | Please Do Not Staple | Fold and tape

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

International Business Machines Corporation
Department D58, Building 706-2
PO Box 390
Poughkeepsie, New York 12602

Fold and tape | Please Do Not Staple | Fold and tape

IBM®

GC28-0646-4

OS/VS2 TSO Command Language Reference (S370-39) Printed in U.S.A. GC28-0646-4

OS/VS2 MVS JCL
GC28-0692-4

READER'S COMMENT FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. This form may be used to communicate your views about this publication. They will be sent to the author's department for whatever review and action, if any, is deemed appropriate.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation whatever. You may, of course, continue to use the information you supply.

Note: ***Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.***

Possible topics for comments are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If comments apply to a Selectable Unit, please provide the name of the Selectable Unit _____.

If you wish a reply, give your name and mailing address:

_____

_____

_____

Note: Staples can cause problems with automated mail sorting equipment. Please use pressure sensitive or other gummed tape to seal this form. — Cut or Fold Along Line

Please circle the description that most closely describes your occupation.

Customer	(Q) Install Mgr.	(U) System Consult.	(X) System Analyst	(Y) System Prog.	(Z) Applica. Prog.	(F) System Oper.	(I) I/O Oper.	(L) Term. Oper.	(O) Other

IBM	(S) System Eng.	(P) Prog. Sys. Rep.	(A) System Analyst	(B) System Prog.	(C) Applica. Prog.	(D) Dev. Prog.	(R) Comp. Prog.	(G) System Oper.	(J) I/O Oper.	(E) Ed. Dev. Rep.	(N) Cust. Eng.	(T) Tech. Staff Rep.

Number of latest Newsletter associated with this publication: _____

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.)

GC28-0692-4

Reader's Comment Form

Cut or Fold Along Line

Fold and tape | Please Do Not Staple | Fold and tape

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

POSTAGE WILL BE PAID BY ADDRESSEE:

International Business Machines Corporation
Department D58, Building 706-2
PO Box 390
Poughkeepsie, New York 12602

Fold and tape | Please Do Not Staple | Fold and tape

IBM®

OS/VS2 MVS JCL (S370-36) Printed in U.S.A. GC28-0692-4