GC28-0646-3
File No. S370-39

Systems

OS/VS2 TSO Command Language Reference

VS2 Release 3.7

IBM

Fourth Edition (January, 1976)

This is a major revision of, and obsoletes, GC28-0646-2 and Technical Newsletters GN28-2597 and GN28-2602. See the Summary of Amendments following the Contents. Changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

This edition applies to release 3.7 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.

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

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

Preface

This publication describes the 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, **TSO Terminal User's Guide,** 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 **TSO Terminals** manual describes how to use the terminals supported by TSO.

The major divisions in this book are:

- Introduction
- Basic Information For Using TSO
- The Commands
- Command Procedure Statements
- 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 Procedure Statements" describes the control statements used in command procedures.

The prerequisite publication for this book is **OS/VS2 TSO Terminal User's Guide,** GC28-0645.

The publications referred to in this book are:

- **OS/MVT and OS/VS2 TSO Terminals**, GC28-6762
- **OS/VS2 Access Method Services**, GC26-3841
- **OS/VS Message Library: VS2 System Messages**, GC38-1002
- **OS/VS2 JCL**, GC28-0692
- **OS/VS2 SPL: System Generation Reference**, GC26-3792
- **OS/VS2 SPL: TSO**, GC28-0629
- **IBM System/370 Reference Summary**, GX20-1850

Contents

Figures

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

Changes have been made throughout this publication to reflect a Service Update -- 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)

Summary of Amendments for GC28-0646-2 As Updated by GN28-2597 OS/VS2 Release 3

Changes to this material reflect corrections, rather than additions to the base publication. The major corrective action is to the "Command Procedures" section.

Summary of Amendments
For GC28-0646-2
OS/VS2 Release 3

The following are changes to existing TSO commands or subcommands for OS/VS2 Release 3:

ALLOCATE

New operands added:
- DSNAME(dsname — list or *)— specifies the name of the data set to be allocated. Same as DATASET.
- BLKSIZE(value) — specifies the average length(in bytes) of the records that will be written to a data set. Same as BLOCK.
- DDNAME(name) — specifies the name to be associated with the data set. Same as FILE.
- MSVGP(identifier) — specifies an installation defined group of MSS volumes to be used for system selection of a volume or volumes which are to be mounted.

EDIT

New operands added:
- BLKSIZE(integer) — specifies the maximum length (in bytes) for blocks of records of a new data set. Same as BLOCK.
- LRECL(integer) — specifies the length of records to be created for a new data set. Same as LINE.

END subcommand of EDIT

New operands added:
- SAVE — specifies that the modified data set is to be saved.
- NOSAVE — specifies that the modified data set is not to be saved.

SAVE subcommand of EDIT

New operands added:
- RENUM — performs a renumbering function on a data set.
- UNNUM — performs a function that removes line numbers from a data set.

SUBMIT Subcommand of EDIT

New operands added:
- * — may be used to represent the data set being edited, and a list of data sets may be specified.

FREE

New operands added:
- DSNAME — specifies one or more data set names that identify the data sets to be freed. Same as DATASET.
- DDNAME — specifies one or more file names that identify the data sets to be freed. Same as FILE.

HELP

New operands added:
- MSGID(list) — specifies that you wish to get additional information about VSBASIC messages whose message identifiers are given in the list.

LISTCAT

New operands added:
- CREATION — specifies that entries are to be listed on condition.
- EXPIRATION — specifies that entries are to be listed on condition.
- HISTORY — specifies that the name, owner identification, creation date, expiration date of the entries are to be listed.

The following are new commands or subcommands and their associated operands.

COPY/MOVE subcommand of EDIT

Operands:
- * — Value of the current line pointer.
- LINE1 — specifies the first line of a range to be copied.
- LINE2 — specifies the last line of a range to be copied.
- LINE3 — specifies the beginning line number where the copied data set will be positioned.
- STRING — specifies a sequence of characters representing the first line of a range.
- COUNT — specifies the total number of lines to be copied.
- LINE4 — specifies the relative line number at which the data will be placed for NONUM data sets. For NUM data sets, LINE4 has the same meaning as LINE3.
- INCR(n) — specifies the increment to be used.

EXEC Subcommand of EDIT

Operands: See the EXEC command for operand explanation.

UNNUM Subcommand of EDIT

No operands: Used to delete line numbers from a data set containing line numbers.

The following changes apply to this book for OS/VS2 Release 3:

- The ACCOUNT and OPERATOR commands and subcommands are documented in **OS/VS2 SPL: TSO.**
- The writing and execution of command procedures (CLIST usage) and extended symbolic variable functions are documented in **OS/VS2 TSO Terminal User's Guide.**

New Command Procedure statements are:

- ATTN
- CLOSFILE
- CONTROL
- DATA-ENDDATA
- DO-WHILE-END
- ERROR
- EXIT
- GETFILE
- GLOBAL
- GOTO
- IF-THEN-ELSE
- OPENFILE
- PROC
- PUTFILE
- READ
- READDVAL
- RETURN
- SET
- TERMIN
- WRITE-WRITENR

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.

> In this manual, references are made to IBM program products in various applications. These references are not intended to state or imply that only the IBM program product mentioned may be used in the given application; any functionally equivalent program may be used instead.

Basic Information For Using TSO

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 the publication, **TSO Terminals**. The examples in this book are addressed to a user of an IBM 2741 Communication Terminal.

Standard Terminal Conventions

Certain conventions apply to all TSO terminals. They are:

- Any lowercase letters you type are interpreted by the system as uppercase letters. For example, if you type in:

```
abcDe8-fg
```

  the system interprets it as:

```
ABCDE8-FG
```

The only exceptions are certain text-handling applications which allow you to type in text with both uppercase and lowercase letters.

- All messages or other output sent to you by the system come out in uppercase letters. The only exception is the output from the special text-handling applications mentioned previously, which comes out both in uppercase and lowercase.

Character and Line Deletion

TSO provides a method for you to correct typing mistakes. You can request that the character you just typed be deleted or that all the preceding characters in the line be deleted. You can define your own character-deletion and line-deletion control characters, or you can use the default characters in the system. For example, if the control characters are the quotation mark (") for deleting the preceding character, and the percent sign (%) for deleting the current line, and you type the following message:

```
first ent%Sect"onft""d ENR"try
```

it is received by the system as:

```
SECOND ENTRY
```

Note that you can use the character-deletion character repetitively (to delete more than one of the preceding characters in the line).

The blank space produced when you hit the space bar is also considered to be a character, and you can delete it using the character-deletion or line-deletion characters. For example, if you type the following line:

```
a b%cd  "E "f
```

it is received by the system as:

```
CD EF
```

Normally, you will use the default characters in the system, (usually the backspace and the attention key). However, you can use the PROFILE command to establish your own character-deletion and line-deletion characters. The PROFILE command is described in the section "Starting and Ending a Terminal Session" in **OS/VS2 TSO Terminal User's Guide.** The ability to change the character-deletion and line-deletion characters is useful when you use more than one type of terminal. For example, any time you have to use a terminal that does not have backspace and attention keys, you can use the PROFILE command to select two other suitable characters as the character-deletion and line-deletion characters.

Line by Line Data Entry

After you type a line and make any necessary corrections, you can enter that line as follows:

- Press the RETURN key on an IBM 2741 Communication Terminal.
- Press the RETURN key on an IBM 1052 Printer-Keyboard. (If the 1052 does not have the automatic EOB feature, hold down the ALTN coding key and press the EOB(s) key.)[1]
- Hold the CTRL key and press the XOFF key on a Teletype[2] terminal.

Note:

1. This manual assumes that you are using an IBM 2741 terminal, and that you must press the RETURN key to enter a line.
2. If you want to enter a *null line*, press the key used to enter a line (RETURN key on the 2741) after entering at least one blank.

You cannot use the character-deletion and line-deletion characters to make corrections to the line after you enter it. If the line you entered was a command, you must use the attention interruption (described later in this

---

[1] For information about the terminal you are using, refer to **TSO Terminals.**
[2] Trademark of the Teletype Corporation.

section) to cancel the line. If the line you entered was data, you can change it by using the EDIT command. See the section "Entering and Manipulating Data" in **OS/VS2 TSO Terminal User's Guide.**

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                       operand-2
               (old data-set-name)           (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 lower case 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 upper case 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 the "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 with an immediate carrier return.

```
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. You can also use the TAB key to enter one or more blanks.

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 continued 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. Stacked items represent alternatives. You should select only one item.
   Example: The representation

   $$\begin{array}{l}\texttt{A}\\ \texttt{B}\\ \texttt{C}\end{array}$$

   indicates that A or B or C is to be selected.
5. Hyphens join lower-case words and symbols to form a *single* variable.
   Example: If member-name appears in a command definition, you should substitute a specific value (for example, BETA) for the variable in the actual command.
6. An underscore indicates a default option. If you select an underscored alternative, you need not specify it when you enter the command.
   Example: The representation

   $$\begin{array}{l}\texttt{A}\\ \underline{\texttt{B}}\\ \texttt{C}\end{array}$$

   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.
7. Braces group related items, such as alternatives.
   Examples: The representation

   $$\texttt{ALPHA=(}\left\{\begin{array}{c}\texttt{A}\\ \texttt{B}\\ \texttt{C}\end{array}\right\}\texttt{,D)}$$

   indicates that you must choose one of the items enclosed within the braces. If you select A, the result is ALPHA=(A,D).
8. Brackets also group related items; however, everything within the brackets is optional and may be omitted.
   Example: The representation

   $$\texttt{ALPHA=(}\left[\begin{array}{c}\texttt{A}\\ \texttt{B}\\ \texttt{C}\end{array}\right]\texttt{,D)}$$

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

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

   indicates that ALPHA can appear alone or can be followed by ,BETA any number of times in succession.
10. Stacked items - alternatives; specify only one item from the stack.

    Note: Items in a stack may be either simple (single-line) or complex (multiple-choice).

    Example:

    $$\begin{array}{l}\left\{\begin{array}{c}\texttt{aa}\\ \texttt{bb}\end{array}\right\}\\ \\ \quad\texttt{cc}\\ \\ \left\{\begin{array}{c}\texttt{dd}\\ \texttt{ee}\\ \texttt{ff}\end{array}\right\}\end{array}$$

    represents a three-item stack.

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

The Attention Interruption

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

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

Note: One output record from the interrupted program may be printed 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 (the ATTN key on IBM 2741 Communication Terminals). Whether or not your terminal has a key for attention interruptions, you can use the TERMINAL command to specify particular operating conditions that the system is to interpret as a request for an attention interruption. More specifically, you can specify a sequence of characters that the system is to interpret as a request for an attention interruption. In addition, you can request the system to pause after a certain number of seconds of processing time has elapsed or after a certain number of lines of output has been displayed at your terminal. When the system pauses, you can enter the sequence of characters that you define as a request for an attention interruption.

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

These are three types of responses to an attention interruption entered by a terminal user:

System Response	Explanation
I	Ignored
D	Input line has been deleted.
"attention message"	Indicates mode.

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 prints:

```
READY
```

When you enter a command that has subcommands and the system is ready to accept that command's subcommands, it prints 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 print any output or other messages, as a result of the previous command or TEST subcommand, it does so before printing 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 listing 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 the publication **TSO Terminals** 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 listing 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 RETURN 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 RETURN 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 printed out 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 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 paragraphs:

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:

PARTS.DATA(PART14)
member name

TSO Data Set Names

A data set name must be qualified in order to conform to the TSO data set naming conventions. The qualified name must consist of at least the two required fields of the following three:

1. Your user-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)	—

Figure 2. Default Names Supplied by the System

Note: Member names must be enclosed in parentheses to distinguish them from data set names.

	Descriptive Qualifiers		
Command	**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 may specify the password as part of the data set name (you will be prompted for it otherwise). The password is separated from the data set name by a slash (/) and optionally, by one or more standard 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.**

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}  ({DATASET} { (*)            } [FILE(name)  ] )
{ALLOC   }  ({DSNAME } { (dsname-list)  } [DDNAME(name)] )
            (DUMMY                                       )
            ({FILE(name)  } [DATASET { (*)           }]  )
            ({DDNAME(name)} [DSNAME  { (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(userid)]
            [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 the data set.
You may substitute an asterisk (*) for the data set name to indicate that you want to have your terminal allocated for input and output. If you use an asterisk (*), only the FILE 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: 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.

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

1. 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.
2. 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.
3. 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.

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 filename FILE"; for instance, DCL MASTER FILE. For COBOL, this name is the external-name used in the ASSIGN TO clause. For FORTRAN, this name is the data set reference number that identifies a data set and has the form "FTxxFyyy;" for instance FT06F002.
If you omit this operand, the system assigns an available file name (ddname) from a data definition statement in the procedure that is invoked when you enter the LOGON command.

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

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 eight 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 the data set is not to be placed on a HOLD queue upon unallocation. NOHOLD is the default if neither HOLD nor NOHOLD is specified.

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.

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**.

Note: 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.

```
{ATTRIB}   attr-list-name
{ATTR  }
           [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,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.

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($\left\{\begin{matrix}\textbf{F}\\\textbf{D}\end{matrix}\right\}$ **)**

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,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.
- **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-** accept the block of records in which the error was found.
- **SKP-** skip the block of records in which the error was found.
- **ABE-** end the task abnormally.

BFTEK({S | E | A | R})

specifies the type of buffering that you want the system to use. The types that you can specify are:

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

RECFM(A,B,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)
```

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.

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.

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

```
{DELETE}     (entryname[/password] [...])
{DEL   }     [CATALOG(catname[/password])]
             [FILE(ddname)]
             [ {PURGE}       ]
             [ {PRG  }       ]
             [               ]
             [ {NOPURGE}     ]
             [ {NPRG   }     ]
             [  ERASE        ]
             [ {NOERASE}     ]
             [ {NERAS  }     ]
             [  SCRATCH      ]
             [ {NOSCRATCH}   ]
             [ {NSCR     }   ]
             [  CLUSTER                      ]
             [ {USERCATALOG}                 ]
             [ {UCAT       }                 ]
             [ {SPACE}                       ]
             [ {SPC  }                       ]
             [ {NONVSAM}                     ]
             [ {NVSAM  }                     ]
             [  ALIAS                        ]
             [ {GENERATIONDATAGROUP}         ]
             [ {GDG                }         ]
             [ {PAGESPACE}                   ]
             [ {PGSPC    }                   ]
```

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.

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 VSAM 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 VSAM entry is not 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:
Your userid is D27UCAT1

```
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 D27UCAT1.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.

EDIT Command

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.

```
{EDIT}    data-set-name[/password]
{E   }
          [NEW]
          [OLD]
          [PLI     [([integer 1] [integer 2]] [CHAR60])]
                     [   2     ] [   72    ]] [CHAR48])]
          [PLIF    [([integer 1] [integer 2]] [CHAR60])]
                     [   2     ] [   72    ]] [CHAR48])]
          [ASM                                          ]
          [COBOL                                        ]
          [GOFORT [FREE ]                               ]
          [       [FIXED]                               ]
          [FORTGI                                       ]
          [FORTH                                        ]
          [TEXT                                         ]
          [DATA                                         ]
          [CLIST                                        ]
          [CNTL                                         ]
          [VSBASIC                                      ]
          [SCAN  ]
          [NOSCAN]
          [NUM  ]  [(integer1[integer2])]
          [NONUM]
          [BLOCK(integer)  ]
          [BLKSIZE(integer)]
          [LINE(integer) ]
          [LRECL(integer)]
          [CAPS]
          [ASIS]
```

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, the sequential data set for you when you save it. If you specify NEW and include a member name the system allocates a partitioned data set and creates the indicated member when you try to save it.

OLD specifies that the data set named on the EDIT command already exists. When you specify OLD and the system is unable to locate the data set, you will be notified and you will have to reenter the EDIT command. If you specify OLD without specifying a member name, the system will assume that your data set is sequential: if the data set is in fact a partitioned data set, the system will assume that the member name is TEMPNAME. If you specify OLD and include a member name, the system will notify you if your data set is not partitioned.

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

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

CHAR48 or CHAR60 CHAR48 specifies that the PL/I source statements are written using the character set that consists of 48 characters. CHAR60 specifies that the source statements are written using the character set that consists of 60 characters. If 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 FORT 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 (i.e., it is enclosed in quotes).
- The data-set-name operand is a member name only (i.e., it is enclosed in parentheses).
- The data-set-name operand does not include a descriptive qualifier; or the descriptive qualifier is such that EDIT cannot determine the data set type. (See Figure 3 for a list of valid descriptive qualifiers.)

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

Note: When the descriptive qualifier FORT is entered with no data set type, the data set type default is GOFORT(FREE). If PLI is the descriptive qualifier, the data set type default is PLI. To use data set types

GOFORT(FIXED), FORTGI, FORTH or PLIF, 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.

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 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 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 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	CAPS/ASIS	
		LINE(n)		BLOCK(n)		NUM (n, m)		
		default	specif	default	specif. (Note 1)	default (n, m) spec.	default	CAPS Required
ASM	PS/PO	80	=80	3120	≤default	Last 8 73≤n≤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,				400				
FORTH,				400				
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.

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

Input Mode

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

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

Line Numbers: Unless you specify otherwise, the system assigns a line number to each line as it is entered. 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. More specific considerations are:

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

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 colums 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 (i.e., a hyphen or plus sign followed immediately by a carrier return). 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 (e.g., CHANGE, INSERT) to enter data, a hyphen at the end of the line indicates subcommand continuation; the system will append the continuation data to the subcommand.

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

Edit Mode

You can enter subcommands to edit data sets when you are in edit mode. You can edit data sets that have line numbers by referring to the number of the line that you want to edit. This is called *line-number editing*. You can also edit data by referring to specific items of text within the lines. This is called *context editing*. A data set having no line numbers may be edited only by context. Context editing is performed by using subcommands that refer to the current line value or a character combination, such as with the FIND or CHANGE subcommands. There is a pointer within the system that points to a line within the data set. Normally, this pointer points to the last line that you referred to. You can use subcommands to change the pointer so that it points to any line of data that you choose. You may then refer to the line that it points to by specifying an asterisk (*) instead of a line number. Figure 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 carrier return key without typing anything first.

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

- 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), it 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 carrier return key without typing anything first.
- You cause an attention interruption.
- There is no more space for records to be inserted into the data set and resequencing is not allowed.
- When an error is discovered by the syntax checker.

Data Set Disposition

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

Tabulation Characters

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

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

Executing User Written Programs

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

Terminating the EDIT Command

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

$$\left\{\begin{array}{l}\text{BOTTOM}\\\text{B}\end{array}\right\}$$

CHANGE Subcommand of EDIT

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

```
{CHANGE}   [*                                ]
{C     }   [line-number-1 [line-number-2]    ]
           [*[count 1]                       ]
           {string1 [string2 [ALL]]}
           {count2                 }
```

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

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

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

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

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

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.

Note: If you cause an attention interuption 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.

Note: 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, e.g., '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, e.g., 'pilgrim''s progress'.
- A null string is represented by two single quotes, e.g., ''.

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

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

*—using the ! 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 hit attention 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 listed at your terminal. You complete the line by typing the new information and enter the line by pressing the RETURN key. All of your changes will be incorporated into the data set.

Example 4

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

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

```
change 103 m315 h.p. at 2400
```

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

Example 5

Operation: Change the values in a table.

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

```
change 387 406 52
```

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

Example 6

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

Known:
The sequence of characters: in the beginning

```
change * //in the beginning
```

Example 7

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

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

```
change 15 /weekly//          or       change 15 /weekly/
```

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 towards 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.

```
{COPY}   [ line1      [line2]   [line3]   [INCR(lines)] ]
{CO  }   [                      [  *  ]                 ]
         [ {'string'} [count]   [line4]   [INCR(lines)] ]
         [ {   *    } [  1  ]   [  *  ]                 ]
```

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: 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	KKKKKKKKKKK	or			0100	JJJJJJJJJJ
0120	LLLLLLLLLLLL				0110	KKKKKKKKKKK
		copy			0120	LLLLLLLLLLLL
		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
JJJJJJJJJ				IIIIIIIII
KKKKKKKKKK	or			JJJJJJJJJ
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:		*Enter:*		*After:*	
0010	A	copy 60 50		0010	A
0020	BB			0020	BB
0030	CCC			0030	CCC
0040	DDDD			0040	DDDD
0050	EEEEE			0050	EEEEE
0060	FFFFFF		CLP	0060	FFFFFF
0070	GGGGGGG			0061	FFFFFF
0080	HHHHHHHH			0070	GGGGGGG
0090	IIIIIIIII			0080	HHHHHHHH
0100	JJJJJJJJJJ			0090	IIIIIIIII
0110	KKKKKKKKKKK			0100	JJJJJJJJJJ
0120	LLLLLLLLLLLL			0110	KKKKKKKKKKK
				0120	LLLLLLLLLLLL

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:	*Enter*		*After:*
NOW IS	top		NOW IS
THE TIME	copy 'men' 99999999		THE TIME
FOR ALL			FOR ALL
GOOD MEN			GOOD MEN
TO COME			TO COME
TO THE			TO THE
AID OF			AID OF
THEIR			THEIR
COUNTRY			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:		*Enter:*		*After:*	
0010	A	copy 10 30 100 incr(5)		0010	A
0020	BB			0020	BB
0030	CCC	or		0030	CCC
0040	DDDD			0040	DDDD
0050	EEEEE	copy 9 31 100 incr(5)		0050	EEEEE
0060	FFFFFF			0060	FFFFFF
0070	GGGGGGG	or		0070	GGGGGGG
0080	HHHHHHHH			0080	HHHHHHHH
0090	IIIIIIIII	copy 1 39 100 incr(5)		0090	IIIIIIIII
0100	JJJJJJJJJJ			0100	JJJJJJJJJJ
0110	KKKKKKKKKKK			0105	A
0120	LLLLLLLLLLLL			0110	BB
			CLP	0115	CCC
				0116	KKKKKKKKKKK
				0120	LLLLLLLLLLLL

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:		***Enter:***		***After:***	
0010	A	copy 20 50 50		0010	A
0020	BB			0020	BB
0030	CCC			0030	CCC
0040	DDDD			0040	DDDD
0050	EEEEE			0050	EEEEE
0060	FFFFFF			0060	BB
0070	GGGGGGG			0070	CCC
0080	HHHHHHHH			0080	DDDD
0090	IIIIIIIII		CLP	0090	EEEEE
0100	JJJJJJJJJJ			0091	FFFFFF
0100	KKKKKKKKKKK			0092	GGGGGGG
0120	LLLLLLLLLLLL			0093	HHHHHHHH
				0094	IIIIIIIII
				0100	JJJJJJJJJJ
				0110	KKKKKKKKKKK
				0120	LLLLLLLLLLLL

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:		***Enter:***		***After:***	
0010	A	copy 70 110 50 incr(10)		0010	A
0020	BB			0020	BB
0030	CCC			0030	CCC
0040	DDDD			0040	DDDD
0050	EEEEE			0050	EEEEE
0060	FFFFFF			0060	GGGGGGG
0070	GGGGGGG			0070	HHHHHHHH
0080	HHHHHHHH			0080	IIIIIIIII
0090	IIIIIIIII			0090	JJJJJJJJJJ
0100	JJJJJJJJJJ		CLP	0100	KKKKKKKKKKK
0110	KKKKKKKKKKK			0101	FFFFFF
0120	LLLLLLLLLLLL			0102	GGGGGGG
				0103	HHHHHHHH
				0104	IIIIIIIII
				0105	JJJJJJJJJJ
				0110	KKKKKKKKKKK
				0120	LLLLLLLLLLLL

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 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	KKKKKKKKKKK
0120	LLLLLLLLLLLL

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	KKKKKKKKKKK
	0120	LLLLLLLLLLLL

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	KKKKKKKKKKK
0120	LLLLLLLLLLLL

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	KKKKKKKKKKK
	0163	LLLLLLLLLLLL

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	KKKKKKKKKKK
	0162	LLLLLLLLLLLL

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.

```
{DELETE}    [ *                             ]
{DEL   }    [ line-number-1 [line-number-2] ]
            [ *[count]                      ]
```

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

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

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

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

Example 1

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

```
delete *
```

or

```
delete
```

or

```
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, new commands may be entered after entering the END subcommand. If you have modified your data set and have not entered the SAVE subcommand or one of the optional keywords, the system will ask you if you want to save the data set. At this point, you may enter the SAVE subcommand. If you do not wish to save the data set, reenter the END subcommand.

```
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.

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}        string [position]
{F   }
```

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. 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 a special delimiter. The extra character may be any printable character other than a number, apostrophe, semicolon, blank, tab, comma, parenthesis, or asterisk. You must not use the extra character in the character string. Do not put a delimiter between the extra character and the string of characters.
Instead of using special delimiters to indicate a character string, you can use paired single quotes (apostrophes) to accomplish the same function with the FIND subcommand. The use of single quotes as delimiters for a character string is called quoted-string notation. Following are the rules for quoted-string notation for the string operand:

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

position specifies the column within each line at which you want the comparison for the string to be made. This operand specifies the starting column of the field to which the string is compared in each line. If you want to consider a string starting in column 6, you must specify the digit 6 for the positional operand. When you use this operand with the special delimiter form of notation for "string", you must separate it from the string operand with the same special delimiter as the one preceeding 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.

```
{INPUT}   [line-number[increment]]
{I    }   [*                     ]

          [R]
          [I]

          [PROMPT  ]
          [NOPROMPT]
```

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

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

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

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

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

NOPROMPT specifies that you do not want to be prompted.

Example 1

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

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

```
input 2 2 r prompt
```

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

```
edit 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 listing at your terminal will resemble the following sample listing:

```
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, FIND and LIST subcommands.

```
{INSERT}        [insert-data]
{IN    }
```

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

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

Example 2

Operation: Insert several lines into a data set.

Known:
The data set contains line numbers.
The inserted lines are to follow line number 00280.
The current line pointer points to line number 00040.
The user is operating in EDIT mode.
The lines to be inserted are:
"J.W. HOUSE 13-244831 24.73"
"T.N. HOWARD 24-782095 3.05"
"B.H. IRELAND 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 listing at your terminal will be similar to the following sample listing:

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

```
*
```

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

```
{LIST}     [ line-number-1 [line-number-2] ]
{L   }     [ *[count]                      ]

           [ NUM  ]
           [ SNUM ]
```

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

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

* specifies that the line referred to by the 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 isn't 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, i.e., not printed on the listing.

Example 1

Operation: List an entire data set.

```
list
```

Example 2

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

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

```
list 27 44
```

Example 3

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

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

```
list * 17
```

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 towards 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.

```
{MOVE}   { line1        [line2]    [line3]  [INCR(lines)] }
{MO  }   {                         [ *   ]                }
         {                                                }
         { {'string'}   [count]    [line4]  [INCR(lines)] }
         { {   *    }   [  1  ]    [ *   ]                }
```

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    KKKKKKKKKKK
0120    LLLLLLLLLLLL
```

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 KKKKKKKKKKK	or	0110 KKKKKKKKKKK
0120 LLLLLLLLLLLL		0120 LLLLLLLLLLLL
	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
KKKKKKKKKKK	or	KKKKKKKKKKK
LLLLLLLLLLLL		LLLLLLLLLLLL
	move 'gg'	

Note that 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:	*Enter:*		*After:*
0010 A	move 60 50		0010 A
0020 BB			0020 BB
0030 CCC			0030 CCC
0040 DDDD			0040 DDDD
0050 EEEEE			0050 EEEEE
0060 FFFFFF		CLP	0060 FFFFFF
0070 GGGGGGG			0070 GGGGGGG
0080 HHHHHHHH			0080 HHHHHHHH
0090 IIIIIIIII			0090 IIIIIIIII
0100 JJJJJJJJJJ			0100 JJJJJJJJJJ
0110 KKKKKKKKKKK			0110 KKKKKKKKKKK
0120 LLLLLLLLLLLL			0120 LLLLLLLLLLLL

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:	*Enter*		*After:*
NOW IS	top		NOW IS
THE TIME	move 'men' 99999999		THE TIME
FOR ALL			FOR ALL
GOOD MEN			TO COME
TO COME			TO THE
TO THE			AID OF
AID OF			THEIR
THEIR			COUNTRY
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:	*Enter:*		*After:*
0010 A	move 10 30 100 incr(5)		0040 DDDD
0020 BB			0050 EEEEE
0030 CCC	or		0060 FFFFFF
0040 DDDD			0070 GGGGGGG
0050 EEEEE	move 9 31 100 incr(5)		0080 HHHHHHHH
0060 FFFFFF			0090 IIIIIIIII
0070 GGGGGGG	or		0100 JJJJJJJJJJ
0080 HHHHHHHH			0105 A
0090 IIIIIIIII	move 1 39 100 incr(5)		0110 BB
0100 JJJJJJJJJJ		CLP	0115 CCC
0110 KKKKKKKKKKK			0116 KKKKKKKKKKK
0120 LLLLLLLLLLLL			0120 LLLLLLLLLLLL

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:	*Enter:*	*After:*
0010 A	move 20 50 50	0010 A
0020 BB		0060 BB
0030 CCC		0070 CCC
0040 DDDD		0080 DDDD
0050 EEEEE		CLP 0090 EEEEE
0060 FFFFFF		0091 FFFFFF
0070 GGGGGGG		0092 GGGGGGG
0080 HHHHHHHH		0093 HHHHHHHH
0090 IIIIIIIII		0094 IIIIIIIII
0100 JJJJJJJJJJ		0100 JJJJJJJJJJ
0110 KKKKKKKKKKK		0110 KKKKKKKKKKK
0120 LLLLLLLLLLLL		0120 LLLLLLLLLLLL

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	*Enter:*	*After:*
0010 A	move 70 110 50 incr(10)	0010 A
0020 BB		0020 BB
0030 CCC		0030 CCC
0040 DDDD		0040 DDDD
0050 EEEEE		0050 EEEEE
0060 FFFFFF		0060 GGGGGGG
0070 GGGGGGG		0070 HHHHHHHH
0080 HHHHHHHH		0080 IIIIIIIII
0090 IIIIIIIII		0090 JJJJJJJJJJ
0100 JJJJJJJJJJ		CLP 0100 KKKKKKKKKKK
0110 KKKKKKKKKKK		0101 FFFFFF
0120 LLLLLLLLLLLL		0120 LLLLLLLLLLLL

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:	*Enter*	*After:*
0010 A	top	0000 HHHHHHHH
0020 BB	move 80 100 * incr(50)	0050 IIIIIIIII
0030 CCC	CLP	0100 JJJJJJJJJJ
0040 DDDD	or	0101 A
0050 EEEEE		0102 BB
0060 FFFFFF	move 80 100 0 incr(50)	0103 CCC
0070 GGGGGGG		0104 DDDD
0080 HHHHHHHH		0105 EEEEE
0090 IIIIIIIII		0106 FFFFFF
0100 JJJJJJJJJJ		0107 GGGGGGG
0110 KKKKKKKKKKK		0110 KKKKKKKKKKK
0120 LLLLLLLLLLLL		0120 LLLLLLLLLLLL

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:	*Enter:*	*After:*
0000 ZIP	top	0050 HHHHHHHH
0010 A	move 80 100 * incr(50)	0100 IIIIIIIII
0020 BB	CLP	0150 JJJJJJJJJJ
0030 CCC	The attempt to move into	0151 ZIP
0040 DDDD	line 0 gets the target data	0152 A
0050 EEEEE	to the top of the data set	0153 BB
0060 FFFFFF	but shifts the target line	0154 CCC
0070 GGGGGGG	by the increment value.	0155 DDDD
0080 HHHHHHHH		0156 EEEEE
0090 IIIIIIIII		0157 FFFFFF
0100 JJJJJJJJJJ		0158 GGGGGGG
0110 KKKKKKKKKKK		0159 KKKKKKKKKKK
0120 LLLLLLLLLLLL		0160 LLLLLLLLLLLL

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. CLP

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

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}     [new-line-no.[increment[old-line-no.[end-line-no.]]]]
{REN  }
```

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

```
{RUN}    ['parameters']
{R  }
         [TEST   ]
         [NOTEST ]
         [LMSG]
         [SMSG]
         [CHECK]
         [OPT  ]
         [LIB(data-set-list)]
         [STORE  ]
         [NOSTORE]
         [GO  ]
         [NOGO]
         [SIZE(value)]
         [PAUSE  ]
         [NOPAUSE]
```

'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.

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

Program Product	Default Operand
Code and Go	SMSG

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

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.

```
{SAVE}    [{  *    }[ RENUM  [(new-line-num[incr[old-line-num  ]]
{S   }    [{dsname }[         [end-line-num]]])]               ]]
          [         [ UNNUM                                    ]]
```

* 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 means, perform a function that will renumber a data set before saving it.

new-line-num 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-num is specified.

old-line-num 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-num must be equal to or less than the new-line-num. This operand cannot be included unless "incr" is specified.

end-line-num 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-num must be greater than the old-line-num. This operand cannot be included unless the old-line-num is specified.

UNNUM means, perform a function which will unnumber a data set before saving it.

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, i.e., the record format will be changed.
- You will be prompted to enter another data set name for SAVE or a carrier return (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-num	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 PL/I(F), or 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. SUBMIT 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.

$$\left\{\begin{matrix}\textbf{SUBMIT}\\ \textbf{SUB}\end{matrix}\right\} \quad \left\{\begin{matrix}\underline{*}\\ \textbf{(data-set-list)}\end{matrix}\right\} \quad \left[\begin{matrix}\underline{\text{NOTIFY}}\\ \text{NONOTIFY}\end{matrix}\right]$$

***** 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.
- SUBMIT does not support job entry subsystem control cards which precede the JOB card.

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.

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 data sets, the logical tab columns apply only to the data that you actually enter. Since a printed line number prompt is not logically part of the data you are entering, the logical tab positions are calculated beginning at the next position after the prompt. Thus, if you are receiving five-digit line number prompts and have set a logical tab in column 10, the mechanical tab should be set 15 columns to the right of the margin. If you are not receiving line number prompts, the mechanical tab should be set 10 columns to the right of the margin.

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

```
{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". 10 settings is the maximum number of tabs allowable. Do not use the tab key to produce the new image line. A good practice is to use a sequence of digits between the "t"s so you can easily determine which columns the tabs are set to. (See Example 3.)

Example 1

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

Known:
Tab settings are not in effect.

```
tab
```

Example 2

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

```
tab on(2 18 72)
```

Example 3

Operation: Establish tabs at every 10th column.

```
tab image
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.

$$\left\{\begin{matrix}\textsf{VERIFY}\\\textsf{V}\end{matrix}\right\} \qquad \left[\begin{matrix}\underline{\textsf{ON}}\\\textsf{OFF}\end{matrix}\right]$$

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 followed by the name of the data set that contains the command procedure.
- *The implicit form:* Do *not* enter EXEC but only enter the name of the 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 implicit procedure name. TSO will only search the SYSPROC file for the specified name. If the procedure resides in SYSPROC, this form is the fastest 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 on "Command Procedures" in this book.

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.

```
{EXEC}          data-set-name
{EX  }          ['value-list']
                [NOLIST]
                [LIST  ]
                [NOPROMPT]
                [PROMPT  ]
          or
[%] procedure-name  [value-list]
```

data-set-name specifies the name of the data set containing the command procedure to be executed. If the descriptive qualifier for the data set is not CLIST, you must enclose the fully qualified name within apostrophes and the data set must contain line numbers according to the following format:

```
Variable blocked - columns 1-8
Fixed blocked - columns  last 8 bytes of each record
```

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

Note: Command procedures are explained in greater detail in the **OS/VS2 Terminal User's Guide.**

procedure-name specifies a member of a command procedure library. The library must previously have been defined with the SYSPROC DD statement of the LOGON procedure or with the ALLOCATE command.

%procedure-name specifies a member of a command procedure library in an extended implicit form that causes an immediate attachment of the EXEC command processor.

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 PL/I compiler.

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

The command procedure consists of:

```
proc 1 name
allocate dataset ( &name..pli) file( sysin)
allocate dataset( &name..list) file( sysprint) block( 80) -
    space( 300,100)
allocate dataset( &name..obj) file( syslin) block( 80) -
    space( 250,100)
allocate file( sysut1) block( 1024) space( 60,60)
allocate file( sysut3) block( 80) space( 250,100)
call 'sys1.linklib( iemaa)' 'list,atr,xref,stmt'
free file( sysut1,sysut3,sysin,sysprint)
```

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

The name of your program is 'EXP'.

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

```
exec plir 'exp' list
```

The listing at your terminal will be similar to:

```
allocate dataset(exp.pli) file(sysin)
allocate dataset(exp.list) file(sysprint) block(80) -
space(300,100)
allocate dataset(exp.obj) file(syslin) block(80) -
space(250,100)
allocate file(sysut1) block(1024) space(60,60)
allocate file(sysut3) block(80) space(250,100)
call 'sys1.linklib(iemaa)' 'list,atr,xref,stmt'
free file(sysut1,sysut3,sysin,sysprint)
ready
```

Example 2

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

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

```
plia exp
```

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, and later freed automatically, by the system's command processors.

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

When 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.

```
FREE      ( DSNAME(dataset-name-list)   )   Note:
          | DATASET(dataset-name-list)  |   Choose
          < DDNAME(file-name-list)      >   one
          | FILE(file-name-list)        |   or
          ( ATTRLIST(attr-list-names)   )   more
          [ [DEST(userid)]   [SYSOUT(class)]  ]
          [ [HOLD  ]         [SYSOUT(class)]  ]
          [ [NOHOLD]                          ]
          [ [KEEP                      ]      ]
          [ [DELETE  SYSOUT(class)     ]      ]
          [ [CATALOG                   ]      ]
          [ [UNCATALOG                 ]      ]
```

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(list-of-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(userid) specifies that the SYSOUT data set is to be routed to the user whose user identification corresponds to that given for "userid." If this keyword is omitted on a FREE command for SYSOUT data sets, the data sets will remain directed to the user specified at the time of allocation.

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

NOHOLD specifies that the data set is not to be placed on the HOLD queue.

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. 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 also be used to get additional information about a VSBASIC message or messages.

```
{HELP}    [(sub)command-name [[[FUNCTION] [SYNTAX]          ]]]
{H   }    [                  [[[OPERANDS [(list)]]          ]]]
          [                  [[ALL]                          ]]
          [                  [[MSGID(list)]                  ]]
```

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

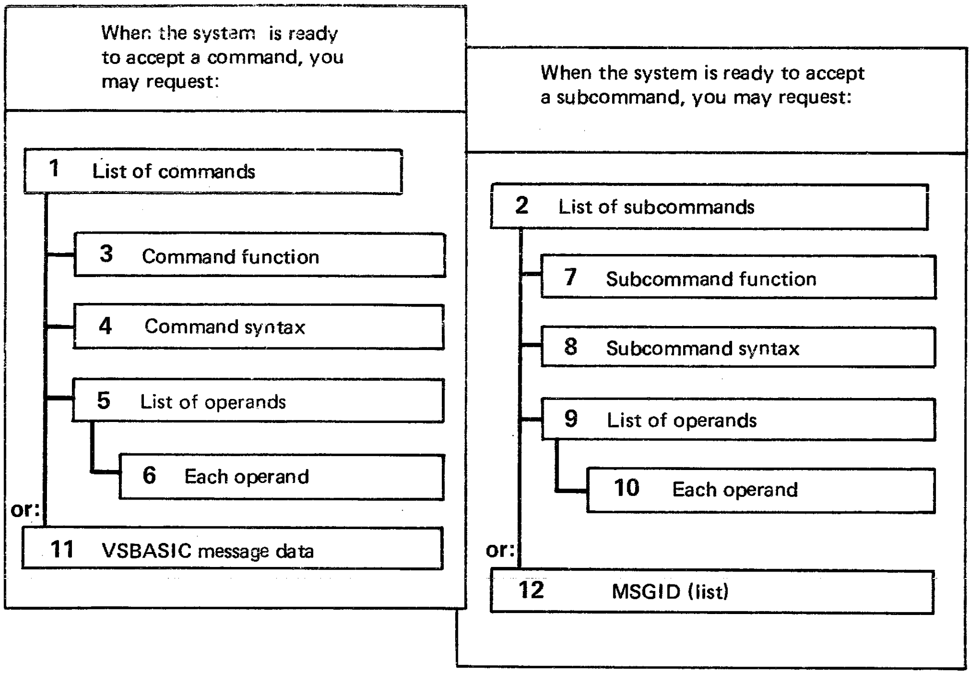
Operation: Have a description of the XREF, MAP, COBLIB, and OVLY operands for the LINK command displayed at your terminal.

```
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

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.

```
LINK      (data-set-list)
          [LOAD[(data-set-name)]]
          [PRINT ({ *             })]
          [       { data-set-name })]
          [NOPRINT                 ]
          [LIB(data-set-list)]
          [PLILIB]        [REFR   ]        [TERM  ]
                          [NOREFR ]        [NOTERM]
          [PLICMIX]
          [PLIBASE]       [SCTR   ]        [DCBS(blocksize)]
                          [NOSCTR ]
          [FORTLIB]
                          [OVLY   ]        [AC(authorization-
          [COBLIB]        [NOOVLY ]             code)]
          [MAP   ]        [RENT   ]
          [NOMAP ]        [NORENT ]
          [NCAL   ]       [SIZE(integer1 integer2)]
          [NONCAL ]
                          [NE  ]
          [LIST   ]       [NONE]
          [NOLIST ]
                          [OL  ]
          [LET    ]       [NOOL]
          [NOLET  ]
                          [DC  ]
          [XCAL   ]       [NODC]
          [NOXCAL ]
                          [TEST  ]
          [XREF   ]       [NOTEST]
          [NOXREF ]
          [REUS   ]
          [NOREUS ]
```

(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 load 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.

PLILIB specifies that the partitioned data set named SYS1.PL1LIB is to be searched by the linkage editor to located 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 located 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 located 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 (E).

NODC specifies that the load module cannot be reprocessed by the linkage editor (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 listing at your terminal will be:

```
allocate dataset(m33payld.load) file(ld2)
link (m33thrus,*) load(m33perfo(mod2)) print(*) -
    plilib map list
IKJ76080A ENTER CONTROL STATEMENTS
    include ld2(mod1)
    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. Also, this command specifies the number of data sets that the system will allow to be allocated to you dynamically. Included in the number of data sets that the system will allow a user to allocate dynamically, are data sets that had been previously allocated for temporary use by a command processor.

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

Note: 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".

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 paritioned
  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}      [MAIL  ]
{LISTB }      [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.

```
{LISTCAT}   [CATALOG(catname[/password] )]
{LISTC  }
            [OUTFILE(ddname)]
            [OFILE(ddname)  ]

            [ ENTRIES(entryname[/password] [...] ) ]
            [ {LEVEL(level)}                       ]
            [ {LVL(level)  }                       ]
            [CLUSTER]
            [DATA]
            [INDEX]
            [IX   ]
            [SPACE]
            [SPC  ]
            [NONVSAM]
            [NVSAM  ]
            [USERCATALOG]
            [UCAT       ]
            [GENERATIONDATAGROUP]
            [GDG                ]
            [PAGESPACE]
            [PGSPC    ]
            [ALIAS]
            [CREATION(days)]
            [EXPIRATION(days)]
            [ALL       ]
            [NAME      ]
            [VOLUME    ]
            [ALLOCATION]
            [HISTORY   ]
```

(NAME is the default, shown underlined.)

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 listed at the terminal.

ENTRIES(entryname[/password]) | LEVEL(level) 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*.

ENTRIES(entry[/password][...]) 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.

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, 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.

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 *data set name(membername).*
- Creation date, expiration date, and, for non-VSAM only, security attributes.
- File name and disposition.
- Non-VSAM data set control blocks (DSCB).

```
{LISTDS}   (data-set-list)
{LISTD }   [STATUS]
           [HISTORY]
           [MEMBERS]
           [LABEL]
           [CATALOG(cat.-name)]
           [LEVEL]
```

(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.

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.

Example

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

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

```
listds cir
```

The listing produced at your terminal will be similar to the listing shown below.

```
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}    (data-set-list)
{LOAD  }    ['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 be the module being processed.

COBLIB specifies that the partitioned data set named SYS1.COBLIB is to be searched to located load modules referred to by the module being processed.

FORTLIB specifies that the partitioned data set named SYS1.FORTLIB is to be searched to located 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 located 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 main storage will be lost.

Note: 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.

```
LOGOFF        [DISCONNECT]
              [HOLD      ]
```

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

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

*

Example 1

Operation: Terminate your terminal session.

```
logoff
```

LOGON Command

Use the LOGON command to initiate a terminal session. Before you can use the LOGON command, your installation must provide you with certain basic information.

- Your user identification (1-7 characters) and, if required by your installation, a password (1-8 alphameric characters).
- An account number (may 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.

```
LOGON          user-identity [/password]
               [ACCT(account)]
               [PROC(procedure)]
               [SIZE(integer)]
               [NOTICES  ]
               [NONOTICES]
               [MAIL  ]
               [NOMAIL]
               [PERFORM(value)]
               [RECONNECT]
```

user-identity and password specifies your user identification and, if required, a valid 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 omit any part of this operand, the system will prompt you to complete the operand. (Printing is suppressed for some types of terminals when you respond to a prompt for a password.)

ACCT(account) specifies the account number required by your installation. If the UADS contains only one account number for the password that you specify, this operand is not required. If the account number is required and you omit it, the system will prompt you for it.
For TSO, an account number must not exceed 40 characters, and must not contain a blank, tab, quotation mark, apostrophe, semicolon, comma, or line control character. Right parentheses are permissible only when left parentheses balance them somewhere in the account number.

PROC(procedure-name) specifies the name of a cataloged procedure containing the Job Control Language (JCL) needed to initiate your session.

SIZE(integer) specifies the 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.

MAIL specifies that you want messages intended specifically for you to be displayed at your terminal. This is the default value if both MAIL and NOMAIL are omitted.

NOMAIL specifies that you do not want to receive messages intended specifically for you.

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.

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 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.
- 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
Teletype* Model 33	**	**
Teletype* Model 35	**	**

* Trademark of Teletype Corporation.
** Refer to the publication *OS/MVT and OS/VS2 TSO Terminals.*

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. You must specify at least one operand or the system will ignore the command.

```
{PROFILE}   [CHAR ({character})]
{PROF   }   [      {   BS    } ]
            [NOCHAR            ]

            [LINE ({ATTN     })]
            [      {character}]
            [      {CTLX     }]
            [NOLINE            ]

            [PROMPT  ]
            [NOPROMPT]

            [INTERCOM  ]
            [NOINTERCOM]

            [PAUSE  ]
            [NOPAUSE]

            [MSGID  ]
            [NOMSGID]

            [MODE  ]
            [NOMODE]

            [LIST]

            [PREFIX(dsname-prefix)]
            [NOPREFIX             ]

            [WTPMSG  ]
            [NOWTPMSG]
```

CHAR(character) 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.

CHAR(BS) specifies that a backspace signals that the previous character entered should be deleted. This is the default value set when your user profile was created.

NOCHAR specifies that no control character is to be used for character deletion.

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

LINE(ATTN) specifies that an attention interruption is to be interpreted as a line-deletion control character. This is the default value set when your user profile was created.

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) specifies that the X and CCTRL keys (depressed together) on a teletype terminal are to be interpreted as a line-deletion control character. This is the default value set when your user profile was created, if you are operating a teletype terminal.

NOLINE specifies that no line-deletion control character (including ATTN) is recognized.

PROMPT specifies that you want the system to prompt you for missing information. This is the default value set when your user profile was created.

NOPROMPT specifies that no prompting is to occur.

INTERCOM specifies that you are willing to receive messages from other terminal users. This is the default value set when your user profile was created.

NOINTERCOM specifies that you do not want to receive messages from other terminals.

PAUSE specifies that you want the opportunity to obtain additional information when a message is issued at your terminal while a command procedure (see the EXEC command) is executing. After a message that has additional levels of information is issued, the system will display the word PAUSE and wait for you to enter a question mark (?) or a carrier return.

NOPAUSE specifies that you do not want prompting for a question mark or carrier return. 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.

Note: 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.

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 1-7 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 if neither operand is specified.

Example 1

Operation: Establish a complete user profile

Known:

The character that you want to use to tell the system to delete the previous character: #

The indicator that you want to use to tell the system to delete the current line: ATTN.

You want to be prompted.

You do not want to receive messages from other terminals.

You want to be able to get second level messages while a command procedure is executing.

You do not want diagnostic message identifiers.

```
profile char(#) line(attn) prompt nointercom pause
nomsgid
```

Example 2

Operation: Suppose that you have established the user profile in Example 1. The terminal that you are using now does not have a key to cause an attention interrupt. You want to change the line delete control character from ATTN to @ without changing any other characteristics.

```
PROF LINE(@)
```

Example 3

Operation: Establish and use a line-deletion character and a character-deletion character.

Known:
The line-deletion character: &
The character-deletion character: !

```
profile line(&) char(!)
```

Now, if you type:

```
now is the ti&ac!bcg!.
```

and press the carrier return key, you will actually enter:

```
abc.
```

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/VS 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.

```
{PROTECT}   data-set-name
{PROT   }   [ADD (password 2)                     ]
            [REPLACE (password 1   password 2)    ]
            [DELETE (password 1)                  ]
            [LIST (password 1)                    ]
            [PWREAD  ]
            [NOPWREAD]
            [PWWRITE]
            [NOWRITE]
            [DATA('string')]
```

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 protect by a password, its security counter will be set and the password being assigned will be flagged as the control password for the data set. The security counter is not affected when additional passwords are entered.

REPLACE(password1, password2) specifies that you want to replace an existing password, access type, or optional security information. The first value (password1) is the existing password; the second value (password2) is the new password.

DELETE(password1) specifies that you want to delete an existing password, access type, or optional security information.
If the entry being removed is the control entry (see the discussion following these operand descriptions), all other entries for the data set will also be removed.

LIST(password1) specifies that you want the security counter, the access type, and any optional security information in the Password Data Set entry to be displayed at your terminal.

password1 specifies the existing password that you want to replace, delete, or have its security information listed.

password2 specifies the new password that you want to add or to replace an existing password.

PWREAD specifies that the password must be given before the data set can be read.

NOPWREAD specifies that the data set can be read without using a password.

PWWRITE specifies that the password must be given before the data set can be written upon.

NOWRITE specifies that the data set cannot be written upon.

DATA('string') specifies optional security information to be retained in the system. The value that you supply for 'string' specifies the optional security information that is to be included in the Password Data Set entry (up to 77 bytes).

Password Data Set

Before you can use the PROTECT command, a Password Data Set must reside on the system residence volume. The Password Data Set contains passwords and security information for protected data sets. You can use the PROTECT command to display this information about your data sets at your terminal.

The Password Data Set contains a security counter for each protected data set. This counter keeps a record of the number of times an entry has been referred to. The counter is set to 'zero' at the time an entry is placed into the data set, and is incremented each time the entry is accessed.

Each password is stored as part of an entry in the Password Data Set. The first entry in the Password Data Set for each protected data set is called the *control entry*. The password from the control entry must be specified for each access of the data set via the PROTECT command, with one exception: the LIST operand of the PROTECT command does not require the password from the control entry.

If you omit a required password when using the PROTECT command, the system will prompt you for it; and if your terminal is equipped with the 'print-inhibit' feature, the system will disengage the printing mechanism at your terminal while you enter the password in response. However, the 'print-inhibit' feature is not use 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}     old-name   new-name
{REN   }     [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.

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
```

RUN Command

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.

```
{RUN}    data-set-name
{R  }
         ['parameters']

         [ASM[LIB(data-set-list)]                                        ]
         [COBOL[LIB(data-set-list)]                                      ]
         [FORT[LIB(data-set-list)]                                       ]
         [PLI [CHECK] [LIB(data-set-list)]                               ]
         [    [OPT  ]                                                    ]
 Note:   [GOFORT [FIXED] [LMSG]                                          ]
 Choose  [       [FREE ] [SMSG]                                          ]
 only -> [VSBASIC [LPREC] [TEST  ] [GO  ] [STORE  ]                      ]
 one if  [        [SPREC] [NOTEST] [NOGO] [NOSTORE]                      ]
 any.    [        [PAUSE  ] [SOURCE] [SIZE(value)]                       ]
         [        [NOPAUSE] [OBJECT]                                     ]
```

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 compiler is to re-use an old object program. This operand is valid only for VSBASIC.

SIZE(value) specifies the number of thousand-byte blocks of user area where value is an integer of 1-3 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, or GOFORT) that you may specify for the RUN command. If you omit this operand, the system checks the descriptive qualifier of the data set name that is to be executed (see the data set naming conventions for a list of descriptive qualifiers). If the system cannot determine the compiler type from the descriptive qualifier, you will be prompted.

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.

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
```

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.

```
{SEND}   'text'
{SE  }   [ [USER ({userid-list}) ] [NOW  ] [NOWAIT] ]
         [       ({     *     }) ] [LOGON] [WAIT  ] ]
         [                         [SAVE ]          ]
         [ [OPERATOR(2)          ]                  ]
         [ [OPERATOR(route-code) ]                  ]
         [ [CN(console-id)]                         ]
```

'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(user-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 alows you to request an attention interruption whether or not your terminal has a key for the purpose.

The terminal characteristics that you have defined will remain in effect until you enter the LOGOFF command. If you terminate a session and begin a new one by entering a LOGON command (instead of a LOGOFF command followed by a LOGON command), the terminal characteristics defined in the earlier session will be in effect during the subsequent session.

```
{TERMINAL}    [LINES(integer)]
{TERM    }    [NOLINES       ]

              [SECONDS(integer)]
              [NOSECONDS       ]

              [INPUT(string)]
              [NOINPUT      ]

              [BREAK  ]
              [NOBREAK]

              [TIMEOUT  ]
              [NOTIMEOUT]

              [LINESIZE(integer)]

              [CLEAR(string)]
              [NOCLEAR      ]

              [SCRSIZE(rows, length)]
```

LINES(integer) specifies an integer from 1 to 255 that indicates you want the opportunity to request an attention interruption after that number of lines of continuous output has been directed to your terminal.

NOLINES(integer) specifies that output line count is not to be used for controlling an attention interruption. This is the default condition.

SECONDS(integer) specifies an integer from 10 to 2550 (in multiples of 10) to indicate that you want the opportunity to request an attention interruption after that number of seconds has elapsed during which the terminal has been locked and inactive. If you specify an integer that is not a multiple of 10, it will be changed to the next largest multiple of 10.

NOSECONDS specifies that elapsed time is not to be used for controlling an attention interruption. This is the default condition.

INPUT(string) specifies the character string that, if entered as input, will cause an attention interruption. The string must be the only input entered and cannot exceed four characters in length.

NOINPUT specifies that no character string will cause an attention interruption. This is the default condition.

BREAK specifies that your terminal keyboard will be unlocked to allow you to enter input whenever you are not receiving output from the system; the system can interrupt your input with high-priority messages. Since use of BREAK with a terminal type which cannot support it can result in loss of output or error, check with your installation system manager before specifying this operand.

NOBREAK specifies that your terminal keyboard will be unlocked only when your program or a command you have used requests input.

Note: The default for the BREAK/NOBREAK operand is determined when your installation defines the terminal features.

TIMEOUT specifies that your terminal's keyboard will lock up automatically after approximately nine to 18 seconds of no input. (Applicable only to the IBM 1052 Printer-Keyboard without the text timeout suppression feature.)

NOTIMEOUT specifies that your terminal's keyboard will not lockup automatically after approximately nine to 18 seconds of no input. (Applicable only to the IBM 1052 Printer-Keyboard with the text timeout suppression feature.)

Note: The default for the TIMEOUT/NOTIMEOUT operand is determined when your installation defines the terminal features.

LINESIZE(integer) specifies the length of the line (the number of characters) that can be printed at your terminal. (Not applicable to the IBM 2260, 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

The integer must not exceed 255.

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

NOCLEAR specifies that you do not want to use a sequence of characters to erase the screen of an IBM 2260, 2265, 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. Valid screen sizes are:

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

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

Example 1

Operation: Modify the characteristics of an IBM 2741 Communication Terminal to allow operation in unlocked-keyboard mode.

Known:

Your terminal supports the break facility. The installation has defined a default of NOBREAK for your terminal.

```
terminal break
```

Example 2

Operation: Modify the characteristics of an IBM 1052 Printer-Keyboard whose attention key cannot be used to interrupt output and whose output line size is greater than 80 characters.

Known:

You want an opportunity to request an attention interruption after ten consecutive lines of output. You want to limit the output line length to 80 characters.

```
terminal lines(10) linesize(80)
```

Example 3

Operation: Establish the characteristics of an IBM 2260 Display Station to allow for attention interruption and screen erasure requests.

Known:

You want an opportunity to request an attention interruption if neither input is requested nor output sent for one minute. You want a $ to stand for an attention interruption request during a regular input operation. You want a % to stand for a screen erasure request.

```
terminal seconds(60) input($) clear(%)
```

TEST Command

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

Note: If the problem program is loaded from a non-LINK library list, a TASKLIB is not set up. Therefore, if the problem program attempts to LOAD, LINK, XCTL, or ATTACH another module from the same library, the module will not be found.

Uses of the TEST Command: Before execution begins you can:

- Supply initial values (test data) that you want to pass to the program.
- Establish breakpoints (after instructions) where execution will be interrupted so that you can examine interim results. (Breakpoints should not be inserted into TSO service routines or into any of the TEST load modules.)

You can then execute the program. When you use the TEST command to load and execute a program, the program must be an object module or a load module suitable for processing. If the program that you want to test is already executing, you can begin testing by interrupting the program with an attention interruption followed by the TEST command with no operands. You can also begin testing after an abnormal ending (ABEND) if the program is still in virtual storage.

Note: If you enter the TEST command without operands, you can test the in-storage copy of your program. If you enter the TEST command with operands, a fresh copy of your program will be brought in for you to test.

Prior to and during execution you can:

- Display the contents of registers and real storage (as when execution is interrupted at a breakpoint).
- Modify the contents of your registers and real 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: Breakpoints will not be honored in any sections of your program when running in privileged mode.

Addressing Conventions Used with TEST: An address used as an operand for a subcommand of TEST may be a symbolic address, a relative address, an absolute address, or a register which may contain an address.

A *symbolic address* consists of one through eight alphameric characters, the first of which is an alphabetic character. The symbolic address must correspond to a symbol in the program that is being tested. Symbols cannot be used if the program being tested is a member of a partitioned data set that is part of a LINK library list unless the partitioned data set is named SYS1.LINKLIB or is the first one in the list, or unless the program is brought into main storage by TEST as an operand of the TEST command or a subsequent load command. A *relative address* is a hexadecimal number preceded by a plus sign (+). An *absolute address* is a hexadecimal number followed by a period.

Address Modifiers: An expression consisting of one of the above address types followed by a plus or a minus displacement value is also a valid address. The plus or minus displacement value can be expressed in either decimal or hexadecimal notation, as follows:

address +14n	specifies the location that is 14 bytes past that designated by "address."
address +14	specifies the location that is 20 bytes past that designated by "address."

Note: Decimal displacement (either plus or minus) is indicated by the n following the numerical offset.

Qualified Addresses: You can qualify symbolic and relative addresses to indicate that they apply to a particular control section (CSECT). To do this, you precede the address by either the name of the load module and the name of csect or just the name of csect. The qualified address must be in the form:

```
.csectname.address
```

or

```
loadname.csectname.address
```

For instance, if the user supplied name of the load module is OUTPUT, the name of the csect is CTSTART, and the symbolic address is TAXRTN you would specify:

```
.ctstart.taxrtn
```

or

```
output.ctstart.taxrtn
```

If you do not include qualifiers, the system assumes that the address applies to the current control section.

General Registers: You can refer to a general register using the COPY, LIST or Assignment of Values subcommands by specifying a decimal integer followed by an R. The decimal integer indicates the number of the registers and must be in the range zero through 15. The contents of the registers are hexadecimal characters. Other references to the general registers imply indirect addressing. The term indirect general register is used to refer to the general registers when they are used for indirect addressing.

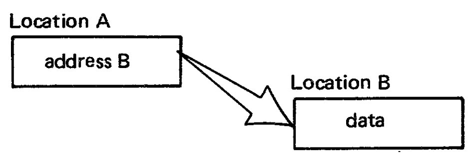
Note: If your program issues the STIMER macro or involves asynchronous interruptions, the contents of your registers may be changed by interruptions even though you are in TEST subcommand mode and your program does not get control.

Floating-Point Registers: You can refer to a floating-point register using the COPY, LIST or Assignment of Values subcommand by specifying a decimal integer followed by an E or a D. An E indicates a floating-point register with a single precision. A D indicates a floating-point register with double precision. The decimal integer indicates the number of the register and must be a zero, two, four, or six. *You must not use floating-point registers for indirect addressing;* expressions composed of references to floating-point registers followed by a plus or minus displacement value or a percent sign are invalid.

Indirect Addresses: An indirect address is an address of a location or general register that contains another address. An indirect address must be followed by a percent sign (the percent sign indicates that the address is indirect). For instance, if you want to refer to some data and the address of the data is located at address A, you can specify:

```
A%
```

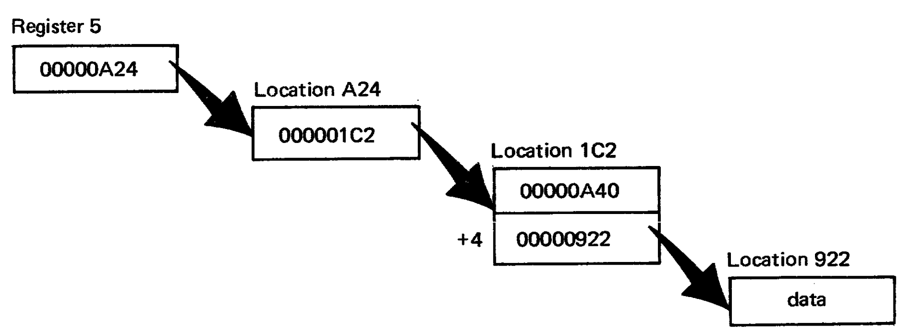
Graphically, this expression indicates:



You can indicate several levels of indirect addresses (256 levels are permitted) by following the initial indirect address with a corresponding corresponding number of percent signs. You can also include plus or minus displacement values. For instance, you may specify:

```
5R%%+4%
```

Graphically, this expression indicates:



Restriction on Symbol Use: You can refer to external symbols in a Load Module if:

- A composite external symbol dictionary (CESD) record exists.
- The TEST operand of the LINK command was specified.

- The program was brought into real storage by the TEST command or one of its subtasks.

You can refer to external symbols in an object module if there is room in real storage for a CESD to be built.

You can refer to most internal symbols if you specify the TEST operand when you assemble and link edit your program. Exceptions are:

- Names on equate statements.
- Names on ORG, LTORG, and CNOP statements.
- Symbols more than eight bytes long.

```
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 in object module form or load module form.

Caution: The program to be tested should not have the name TEST.

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 real storage and in registers.

AT establishes breakpoints at specified locations.

CALL initializes registers and initiates processing of the program at a specified address.

COPY moves data or addresses.

DELETE deletes a load module.

DROP removes symbols established by the EQUATE command from the symbol table of the module being tested.

END terminates all operations of the TEST command and the program being tested.

EQUATE adds a symbol to the symbol table and assigns attributes and a location to that symbol.

FREEMAIN frees a specified number of bytes of real storage.

GETMAIN acquires a specified number of bytes of real 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 real 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 real storage map.

LISTPSW displays the Program Status Word (PSW).

LISTTCB lists the contents of the Task Control Block (TCB) (you may specify the address of another TCB).

LOAD loads a program into real storage for execution.

OFF removes breakpoints.

QUALIFY establishes the starting or base location for relative addresses; resolves identical external symbols within a load module.

RUN terminates TEST and completes execution of the program.

WHERE displays the real address of a symbol or entrypoint or the address of the next executable instruction.

Example 1

Operation: Enter TEST mode after experiencing either an abnormal termination of your program or an interruption.

Known:

Either you have received a message saying that your foreground program has terminated abnormally, or, you have struck the attention key while your program was executing. In either case, you would like to begin "debugging" your program.

```
test
```

Example 2

Operation: Invoke a program for testing.

Known:
The name of the data set that contains the program: TLC55.PAYER.LOAD(THRUST)
The program is a load module and is not a command processor.
The parameters to be passed: 2048, 80

```
test payer(thrust) '2048,80'
```

Example 3

Operation: Invoke a program for testing.

Known:
The name of the data set that contains the program: TLC55.PAYLOAD.OBJ
The program is an object module and is not a command processor.

```
test payload object
```

Example 4

Operation: Test a command processor.

Known:
The name of the data set containing the command processor: TLC55.CMDS.LOAD(OUTPUT)

```
test cmds(output) cp
```

When processing is halted at a breakpoint or before execution is initiated, you can modify values in real 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. The address may be a symbolic address, a relative address, an absolute address, or a register.

data-type 'value' specifies the type of data and the value that you want to place in the specified location. You indicate the type of data by one of the following codes:

Code	**Type of Data**	**Maximum Length (Bytes)**
C	Character	One line of input, continued lines permitted
X	Hexadecimal	64
B	Binary	64
H	Fixed point binary (halfword)	6
F	Fixed point binary (fullword)	11
E	Floating point (single precision)	9
D	Floating point (double precision)	18
L	Extended floating point	16
P	Packed decimal	32
Z	Zoned decimal	17
A	Address constant	10
S	Address (base + displacement)	8
Y	Address constant (halfword)	5

You include your data following the code. Your data must be enclosed within apostrophes. Any single apostrophes within your data must be coded as two single apostrophes. Character data will be entered as is; all other data types will be translated into 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.

Example 1

Operation: Insert a character string at a particular location in real 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'
```

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: A breakpoint should not be established at:

1. The target of an execute instruction.
2. An instruction that will be modified by the execution of other in-line code prior to the execution of the breakpoint.

```
AT        { address[:address] }
          { (address-list)    }
          [(subcommands-list)]
          [COUNT(integer)]
          [NODEFER]
          [DEFER  ]
          [NOTIFY  ]
          [NONOTIFY]
```

(NODEFER and NOTIFY are the underlined defaults.)

address specifies a location that is to contain a breakpoint. The address may be a symbolic address, a relative address, or a general register containing an address. The address must be on a halfword boundary and contain a valid op code.

address:address specifies a range of addresses that are to contain breakpoints. Each address may be a symbolic address, a relative address, an absolute address, or a general register containing an address. Each address must be on a halfword boundary. A breakpoint will be established at each instruction between the two addresses. When a range of addresses is specified, assignment of breakpoints halts when an invalid instruction is encountered.

address-list specifies several addresses that are to contain breakpoints. Each address may be a symbolic address, a relative address, an absolute address, or a general register containing an address. The first address must be on a halfword boundary. The list must be enclosed within parentheses, and the addresses in the list must be separated by standard delimiters (one or more blanks or a comma). A breakpoint will be established at each address.

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 (for instance, LISTTCB PRINT (TCBS);LISTPSW;GO CALCULAT). The list cannot be longer than 255 characters.

COUNT(integer) specifies that processing will not be halted at the breakpoint until it has been encountered a number of times. This operand is directly applicable to program loop situations, where an instruction is executed several times. The breakpoint will be observed each time it has been encountered the number of times specified for the 'integer' operand. The integer specified cannot exceed 65,535.

DEFER specifies that the breakpoint is to be established in a program that is not yet in real storage. The program to contain the breakpoint will be brought in as a result of a LINK, LOAD, ATTACH, or XCTL macro instruction by the program being tested. You must qualify the address of the breakpoint (either LOADNAME.CSECTNAME. RELATIVE or LOADNAME.CSECTNAME.SYMBOL) when you specify this operand. All breakpoint addresses listed in an AT subcommand with the DEFER operand must refer to the same load module.

Note: If the load module is currently in real storage, the defered breakpoints will not be set.

NODEFER specifies that the breakpoint is to be inserted into the program now in real storage. This is the default value if both DEFER and NOFDEFER are omitted.

NOTIFY specifies that when it is encountered the breakpoint will be identified at the terminal. This is the default value if both NOTIFY and NONOTIFY are omitted.

NONOTIFY specifies that when it is encountered the breakpoint will not be identified at the terminal.

Example 1

Operation: Establish breakpoints at each instruction in a section of the program that is being tested.

Known:
The addresses of the first and last instructions of that section that is to be tested: LOOPA EXITA
The subcommands to be executed are: LISTPSW, GO

```
at loopa:exita (listpsw;go)
```

Example 2

Operation: Establish breakpoints at several locations in a program.

Known:
The addresses for the breakpoints: +8A LOOPB EXITB

```
at (+8A loopb exitb)
```

Example 3

Operation: Establish a breakpoint at a location in a loop. The address of the location is contained in register 15. You only want to have an interruption every tenth cycle through the loop.

Known:

The address for the breakpoint: 15R%

```
at 15r% count(10)
```

Example 4

Operation: Establish a breakpoint for a program other than the one presently in real storage.

Known:
The csect name: WIND
The name of the load module: MARCH
The symbolic address for the breakpoint: PROG

```
at march.wind.prog
```

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. The address may be a symbolic address, a relative address, an absolute address, or a register containing an address. Register 15 contains this address when the program under test begins execution.

PARM(address-list) specifies one or more addresses that point to data to be used by the program being tested. The list of addresses will be expanded to fullwords and placed into contiguous storage. Register 1 will contain the address of the start of the list. If PARM is omitted, register 1 will point to a fullword that contains the address of a halfword of zeroes.

VL specifies that the high order bit of the last fullword of the list of addresses pointed to by general register one is to be set to one.

RETURN(address) specifies that register 14 is to contain the address that you supply as the value for this keyword. After the program executes, the system will return control to the point indicated by register 14. If RETURN is omitted, register 14 will contain the address of a breakpoint instruction.

Example 1

Operation: Initiate execution of the program being tested at a particular location.

Known:
The starting address: +0A
The addresses of data to be passed: CTCOUNTR LOOPCNT TAX

```
call +0a parm(ctcountr loopcnt tax)
```

Example 2

Operation: Initiate execution at a particular location.

Known:

The starting address: STARTBD

The addresses of data to be passed: BDFLAGS PRFTTBL COSTTBL ERREXIT

Set the high order bit of the last address parameter to one so that the program can tell the end of the list. After execution, control is to be returned to: +24A

```
call startbd parm(bdflags prfttbl costtbl errexit)-
vl return(+24a)
```

COPY Subcommand of TEST

Use the COPY subcommand to transfer data or addresses from one real storage address to another, from one general register to another, from a register to real storage, or from real 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.

```
{COPY}      address 1   address 2
{C   }
            [LENGTH( integer )]
            [       (   4    )]

            [POINTER  ]
            [NOPOINTER]
```

---

address1 specifies a location that contains data to be copied. The address may be a symbolic address, a relative address, an absolute address, an indirect address, or a qualified address.

address2 specifies a location that will receive the data after it is copied. The address may be a symbolic address, a relative address, an absolute address, an indirect address, or a qualified address.

LENGTH(integer) specifies the length, in decimal, of the field to be copied. If an integer is not specified, LENGTH will default to 4 bytes. The LENGTH keyword can also be entered in the shorter form, L(integer).

POINTER specifies that address1 will be validity checked to see that it does not exceed maximum real 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. The POINTER keyword can also be entered in the shorter form, P.

NOPOINTER specifies that address1 will be treated as an address. NOPOINTER is the default for POINTER.

Note:

1. The COPY subcommand treats the 16 general registers as contiguous fields. If you have specified that 8 bytes be moved from general register 0 to another location.

```
copy 0r 80060. length(8)
```

The COPY subcommand will move the 4 bytes of register 0 and the 4 bytes of register 1 to real 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 real storage location to another.

Known:
The starting address of the data: 80680
The starting address of where the data is to be: 80685

```
copy 80680. 80685. length(8)
```

Example 2

Operation: Copy the contents of one register into another register.

Known:
The register which contains the data to be copied: 10
The register which will contain the data: 5

```
copy 10r 5r
```

Example 3

Operation: Save the contents of the general registers.

Known:
The first register to be saved: 0
The starting address of the save area: A0200

```
c 0r a0200. l(64)
```

Example 4

Operation: Propagate the value in the first byte of a buffer throughout the buffer.

Known:
The starting address of the buffer: 80680
The length of the buffer: 80 bytes

```
c 80680. 80681. l(79)
```

Example 5

Operation: Insert a hexadecimal value into the high-order byte of a register.

Known:
The desired value: X'80'
The register: 1

```
copy 80. 1r l(1) pointer
```

Example 6

Operation: Insert the entry point of a routine into a real storage location.

Known:
The module name and the entry point name: IEFBR14.IEFBR14
The desired real 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 a load module awaiting execution.

```
{DELETE}        load-name
{DEL   }
```

load name specifies the name of the load module to be deleted. The load name is the name by which the program is known to the system when it is in real storage. The name must not exceed eight characters.

Example 1

Operation: The program being tested has called a subroutine that is in load module form. Before executing the subroutine, a breakpoint is encountered. You do not want to execute the subroutine because you intend to pass test data to the program instead. You now want to delete the subroutine since it will not be used.

Known:
The name of the subroutine (load module): TOTAL

```
delete total
```

or

```
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; 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 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

EQUATE Subcommand of TEST

Use the EQUATE subcommand to add a symbol to the symbol table of the module being tested. This subcommand allows you to establish a new symbol that you can use to refer to an address or to override an existing symbol to reflect a new address or new attributes. If no symbol table exists, one is created and the specified name is added to it. You can also modify the data attributes (type, length, and multiplicity). The DROP subcommand removes symbols added by the EQUATE subcommand. Symbols established via the EQUATE subcommand are defined for the duration of the TEST session, only.

```
{EQUATE}    symbol  address  data-type
{EQ    }    [LENGTH(integer)]
            [MULTIPLE(integer)]
```

symbol specifies the symbol (name) that you want to have added to the symbol table so that you can refer to an address symbolically. The symbol must consist of one through eight alphameric characters, the first of which is an alphabetic character.

address specifies a symbolic address, a relative address, an absolute address, or a general register containing an address. The address that you specify will be equated to the symbol that you specify.

data-type specifies either the type of data that you want moved into the location specified via the "address" operand, or the characteristics you wish to attribute to the data at the location given by "address." These may or may not be the same as the original characteristics. You indicate the type of data by one of the following codes:

Code	Type of Data	Maximum Length (Bytes)
C	Character	256
X	Hexadecimal	256
B	Binary	256
I	Assembler instruction	256
H	Fixed point binary (halfword)	8
F	Fixed point binary (fullword)	8
E	Floating point (single precision)	8
D	Floating point (double precision)	8
L	Extended floating point	16
P	Packed decimal	16
Z	Zoned decimal	16
A	Address constant	4
S	Address (base + displacement)	2
Y	Address constant (halfword)	2

LENGTH(integer) specifies the length of the data. The maximum value of the integer is 256. If you do not specify the length, the following default values will apply:

Type of Data	Default Length (Bytes)
C,B,P,Z	1
H,S,Y	2
F,E,A,X	4
D	8
I	variable
L	16

MULTIPLE(integer) specifies a multiplicity factor. The multiplicity factor means that one element of the data appears several times in succession; the number of repetitions is indicated by the number specified for "integer." The maximum value of the integer is 256.

Note: If you do not specify any keywords, the defaults are:

```
type - X
multiplicity - 1
length - 4
```

Example 1

Operation: Add a symbolic address to the symbol table of the module that you are testing.

Known:
The symbol: EXITRTN
The address: TOTAL+4

```
equate exitrtn total+4
```

Example 2

Operation: Change the address and attributes for an existing symbol.

Known:
The symbol: CONSTANT
The new address: 1FAA0.
The new attributes: type: C
length: L(8)
multiplicity: M(2)

```
eq constant 1faa0. c m(2) l(8)
```

FREEMAIN Subcommand of TEST

Use the FREEMAIN subcommand to free a specified number of bytes of real storage.

```
{FREEMAIN}    integer  address
{FREE    }    [SP (integer)]
                  (   0   )
```

integer specifies the number of bytes of real storage to be released.

address specifies a symbolic address, a relative address, an absolute address, or a general register containing an address. This address is the location of the space to be freed and must be a multiple of 8 bytes. The LISTMAP subcommand may be used to help locate previously acquired real storage.

SP(integer) specifies the number of the subpool that contains the space to be freed. If you omit this operand, the default value is subpool zero. The integer must be in the range zero through 127.

Example 1

Operation: Free space in real storage that was acquired previously by a GETMAIN subcommand or by a GETMAIN macro instruction in the module being tested.

Known:
The size of the space, in bytes: 500
The absolute address of the space: 054A20
The number of the subpool that the space was acquired from: 3

```
free 500 054a20. sp(3)
```

GETMAIN Subcommand of TEST

Use the GETMAIN subcommand to obtain a specified number of bytes of real storage.

```
{GETMAIN}   integer
{GET    }   [SP (integer)]
                (   0   )
            [EQUATE(name)]
```

EQUATE(name) specifies that the address of acquired real storage is to be equated to the symbol specified by "name."

integer specifies the number of bytes of real storage to be obtained.

SP(integer) specifies the number of a subpool that contains the bytes of real storage that you want to obtain. If you omit this operand, the default value is subpool zero. The integer must be in the range zero through 127.

Example 1

Operation: Get 500 bytes of real 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 breakdown 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.

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 real storage, or the contents of registers, displayed at your terminal or placed into a data set.

```
{LIST}       {address[:address]} data-type
{L   }       {(address-list)   }
             [LENGTH(integer)]
             [MULTIPLE(integer)]
             [PRINT(data-set-name)]
```

address specifies the location of data that you want displayed at your terminal or placed into a data set. The address may be a symbolic address, a relative address, an absolute address, or a general or floating-point register.

address:address specifies that you want the data located between the specified addresses displayed at your terminal or placed into a data set. Each address may be a symbolic address, a relative address, an absolute address, or a general or floating-point register.

(address-list) specifies several addresses of data that you want displayed at your terminal or placed into a data set. The data at each location will be retrieved. Each address may be a symbolic address, a relative address, an absolute address, or a general or floating-point register. The list of addresses must be enclosed within parentheses, and the addresses must be separated by standard delimiters (one or more blanks or a comma).

data-type specifies the type of data that is in the specified location. You indicate the type of data by one of the following codes:

Code	Type of Data	Maximum Length (Bytes)
C	Character	256
X	Hexadecimal	256
B	Binary	256
I	Assembler instruction	256
H	Fixed point binary (halfword)	8
F	Fixed point binary (fullword)	8
E	Floating point (single precision)	8
D	Floating point (double precision)	8
L	Extended floating point	16
P	Packed decimal	16
Z	Zoned decimal	16
A	Address constant	4
S	Address (base + displacement)	2
Y	Address constant (halfword)	2

LENGTH(integer) indicates the length, in bytes of the data that is to be listed. The maximum value for the integer is 256. If you use a symbolic address and do not specify length, the value for the length parameter will be retrieved from the symbol table residing in the user's region. Otherwise, the following default values will apply:

Type of data	Default Length (Bytes)
C,B,P,Z	1
H,S,Y	2
F,E,A,X	4
D	8
I	variable
L	16

When the data type is I, either length or multiple may be specified, but not both. If both are specified, the multiple parameter is ignored but no error message is printed.

MULTIPLE(integer) Used in conjunction with the length operand. Gives the user the following options:

- The ability to format the data to be listed (see Example 3, below)
- A way of printing more than 256 bytes at a time. (The value supplied for "integer" determines how 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 listed. If you use a symbolic address and do not specify MULTIPLE, the value for the MULTIPLE parameter will be retrieved from the symbol table residing in the users region.

PRINT(data-set-name) specifies the name of a sequential data set to which the data is directed (see data set naming conventions). If you omit this operand, the data will go to your terminal.

The data format is blocked variable length records. Old data sets with the standard format and block size are treated as NEW if being opened for the first time, otherwise, they are treated as MOD data sets.

The LIST subcommands of TEST (LIST, LISTDCB, LISTDEB, LISTMAP, LISTPSW, LISTTCB) perform the following functions on each data set they process.

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 above will be 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 an area of real storage.

Known:
The area to be displayed is between: COUNTERA DTABLE
The attributes of the data: C L(130) M(1)
The name for a data set to contain the listed data: DCDUMP

```
list countera:dtable c l(130) m(1) print(dcdump)
```

Example 2

Operation: List the contents of real storage at several addresses

Known:
The addresses: TOTAL1 TOTAL2 TOTAL3 ALLTOTAL
The attributes of the data: F L(3) M(3)

```
l (total1 total2 total3 alltotal) f l(3) m(3)
```

Example 3

Operation: List the first six fullwords in the Communications Vector Table (CVT).

Known:
The absolute address of the CVT: 10.
The user is operating in TEST mode.
The attributes of the data: X L(4) M(6)

Note: First use the QUALIFY subcommand of TEST to establish the beginning of the CVT as a base location for displacement values.

```
qualify 10.%
```

TEST: The system response

```
list +0 l(4) m(6)
```

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

```
+0      00000000
+4      00012A34
+8      00000B2C
+C      00000000
+10     001A0408
+14     00004430
```

Use the LISTDCB subcommand to list the contents of a data control block DCB. You must provide the address of the beginning of the (DCB).

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 a symbolic address, a relative address, an absolute address, or a general register containing an address. The specified address is the address of the DCB that you want displayed. The address must be on a fullword boundary.

FIELD(names) specifies one or more names of the particular fields in the DCB that you want to have displayed at your terminal. The segment name will not be printed when you use this operand. If you omit this operand, the entire DCB will be displayed.

PRINT(data-set-name) specifies the name of the sequential data set to which data is to be directed (see data set naming conventions). If you omit this operand, the data will be displayed at your terminal.
The data format is blocked variable-length records. Old data sets with the standard record format and blocksize are treated as NEW if they are being opened for the first time; otherwise they are treated as MOD data sets.

The specified data set is kept open until:

- 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: 33B4

```
listdcb 33b4.
```

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 a symbolic address, a relative address, an absolute address, or a general register containing an address. The address is the beginning of the DEB, and must be on a fullword boundary.

FIELD(names) specifies one or more names of the particular fields in the DEB that you want to have displayed at your terminal. If you omit this operand, the entire DEB will be listed.

PRINT(data-set-name) specifies the name of the sequential data set to which data is to be directed (see data set naming conventions). If you omit this operand, the data will be displayed at your terminal.

The data format is blocked variable length records. Old data sets with the standard record format and blocksize are treated as NEW if they are being opened for the first time; otherwise they are treated as MOD data sets.

The specified data set is kept open until:

1. The TEST session is ended by a RUN or END subcommand, or
2. A LIST subcommand is entered that specifies a different PRINT data set. In this case, the former data set is closed and the current one opened.

Example 1

Operation: List the entire DEB for the DCB that is named DCBIN.

Known:
The address of the DEB: DCBIN+2C%

```
listdeb dcbin+2c%
```

LISTMAP Subcommand of TEST

Use the LISTMAP subcommand to display a storage map at your terminal. The map identifies the location and assignment of any storage assigned to the program.

All storage assigned to the problem program and its subtasks as a result of GETMAIN requests is located and identified by subpool (0-127). All programs assigned to the problem program and its subtasks are identified by name, size, location, and attribute. Storage assignment and program assignment are displayed by task.

LISTMAP	[PRINT(data-set-name)]

PRINT(data-set-name) specifies the name of the sequential data set to which data is to be directed (see data set naming conventions). If you omit this operand, the data will be displayed at your terminalal.

The data format is blocked variable length records. Old data sets with the standard record format and blocksize are treated as NEW if they are being opened for the first item; otherwise, they are treated as MOD data sets.

The specified data set is kept open until:

- 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 real storage at your terminal.

```
listmap
```

Example 2

Operation: Direct a map of real storage to a data set.

Known:
The name for the data set: ACDQP.MAP.TESTLIST

```
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 a symbolic address, a relative address, an absolute address, or a general register containing an address. The address identifies a particular PSW. If you do not specify an address, you will receive the current PSW for the program that is executing. (See Appendix B for more information about addresses.)

PRINT(data-set-name) specifies the name of the sequential data set to which data is to be directed (see data set naming conventions). If you omit this operand, the data will be displayed at your terminal.
The data format is blocked variable length records. Old data sets with the standard record format and blocksize are treated as NEW if they are being opened for the first time; otherwise, they are treated as MOD data sets.

The specified data set is kept open until:

- 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: Copy the Input/Output old PSW onto a data set.

Known:
The address of the PSW (in hexadecimal): 38.
The name for the data set: 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 a symbolic address, a relative address, an absolute address, or a general register containing an address. The address must be on a fullword boundary. The address identifies the particular TCB that you want to display. If you omit an address, the TCB for the current task is displayed.

FIELD(names) specifies one or more names of the particular fields in the TCB that you want to have displayed. If you omit this operand, the entire TCB will be displayed.

PRINT(data-set-name) specifies the name of the sequential data set to which data is to be directed (see data set naming conventions). If you omit this operand, the data will be displayed at your terminal.
The data format is blocked variable length records. Old data sets with the standard record format and blocksize are treated as NEW if they are being opened for the first time; otherwise, they are treated as MOD data sets.

The specified data set is kept open until:

1. The TEST session is ended by a RUN or 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: Save a copy of the TCB for the current task on a data set.

Known:
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.TESTLIST

```
listtcb addr(mytcb2) field(tcbtio,tcbcmp,tcbgrs)-
print('scott.testlist')
```

LOAD Subcommand of TEST

Use the LOAD subcommand to load a program into real storage for execution.

LOAD	**program-name**

program name specifies the name of a member of a partitioned data set that contains the load module to be tested. (See the data set naming conventions.)

Example 1

Operation: Load a program named ATX03.LOAD(GSCORES)

```
load (gscores)
```

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 may be a symbolic address, a relative address, an absolute address, or a general register containing an address. The address must be on a halfword boundary.

address:address specifies a range of addresses. Each address may be a symbolic address, a relative address, an absolute address, or a general register containing an address. Each address must be on a halfword boundary. All breakpoints in the range of addresses will be removed.

(address-list) specifies the location of several breakpoints that you want to remove. Each address may be a symbolic address, a relative address, an absolute address, or a general register containing an address. Each address must be on a halfword boundary.

Example 1

Operation: Remove all breakpoints in a section of the program.

Known:
The beginning and ending addresses of the section: LOOPC EXITC

```
off loopc:exitc
```

Example 2

Operation: Remove several breakpoints located at different positions.

Known:
The addresses of the breakpoints: COUNTRA COUNTRB EXITA

```
off (countra countrb exita)
```

Example 3

Operation: Remove all breakpoints in a program.

```
off
```

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.

You can specify an address to be used as the base location for subsequent relative addresses. Each time you use the QUALIFY subcommand, previous qualifications are voided.

Symbols that were established by the EQUATE subcommand before you enter QUALIFY are not affected by the QUALIFY subcommand.

```
{QUALIFY}   {address                                           }
{Q      }   {load-module-name[.entryname]  [TCB(address)]}
```

address specifies an absolute, relative or symbolic address.

load-module-name [entry-name] specifies the name by which a load module is known, and an externally referable name within the load module. The two names are separated by a period. The load module name may be the name or an alias of a member of a partitioned data set. The entry name is the symbolic address of an entry point into the specified module. The entry name may be omitted, in which case the first entry point into the specified module will be supplied.

TCB(address) specifies the address of a task control block (TCB). This operand is necessary when programs of the same name are assigned to two or more subtasks and you must establish uniquely which one is to be qualified, or when the load module request block is not in the TCB chain.

Example 1

Operation: Establish a base location for relative addresses to a symbol within the currently qualified program.

Known:
The base address: QSTART

```
qualify qstart
```

Example 2

Operation: Change the base location for symbolic and relative addresses to a different csect in the program.

Known:
The module name: PROFITS
The entry name (csect): SALES
The TCB address: 5R%

```
qualify profits.sales tcb(5R%)
```

Example 3

Operation: Change the base location for relative addresses to an absolute address.

Known:
The absolute address of the new base: 5F820

```
qualify 5f820.
```

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}          [address]
{R  }
```

address specifies a symbolic address, a relative address, an absolute address, or a general register containing an address. Execution will begin at the specified address. If you do not specify an address, execution begins at the last point of interruption or from the entry point if the GO subcommand was not previously specified.

Example 1

Operation: Execute the program to termination from the last point of interruption.

```
run
```

Example 2

Operation: Execute a program to termination from a specific address.

Known:
The address: +A8

```
run +a8
```

Use the WHERE subcommand to obtain the absolute address serving as the starting or base location for the symbolic and relative addresses in the program. Alternately, you can obtain the absolute address of an 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}   { address                                          }
{W    }   { offset                                           }
          { load-module-name[.entry-name[.offset]]           }
```

address specifies a symbolic address, a relative address, an absolute address, or a general register containing an address. When you specify an address as the operand for the WHERE subcommand, you will receive the name of the load module containing the address.

load-module-name[.entry-name] specifies the name by which a load module is known, and an externally referable name within the load module. The two names are separated by a period. The load module name may be the name or an alias of a member of a partitioned data set. The entry name is the symbolic address of an entry point into the specified module. The entry name may be omitted, in which case the first entry point into the specified module will be supplied. When you specify this operand for WHERE, you will receive the real storage address of the load module.

offset specifies a relative address within a CSECT. When you specify this operand, you will receive the address of the offset.

Example 1

Operation: Obtain the real address of the module named CSTART.

```
where cstart
```

Example 2

Operation: Obtain the real address of the CSECT named JULY in the module named NETSALES.

```
where netsales.july
```

Example 3

Operation: Determine in which program an absolute address is located.

Known:
The absolute address: 3E2B8

```
where 3e2b8.
```

Note: You will also get the TCB address and the relative address.

Example 4

Operation: Determine the absolute address of the next executable instruction.

```
where
```

Example 5

Operation: Determine the virtual address of +1CA.

Known:
The CSECT is WHOOPIE.
The module is GOODTIME.

```
Where goodtime.whoopie.+1ca
```

Note: You will also get the TCB address with the virtual address.

TIME Command

Use the TIME command to obtain the following information:

- Cumulative CPU time (from logon)
- Cumulative session time (from logon)
- Service units used
- Local time of day
- 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: 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 control information. Command procedures reside in data sets that are stored and cataloged just as any other data sets except that the descriptive qualifier of the fully qualified data set name is normally CLIST (command list). Command procedures are often referred to as CLIST.

A command procedure may be invoked by issuing the EXEC command or the EXEC subcommand of EDIT. The EXEC command and subcommand also have an implicit entry facility in normal and extended form. On an implicit entry, enter only the procedure name and the value list. On an extended implicit entry, prefix the procedure name with %; this limits the system search to the file called SYSPROC. (See the EXEC command description for information on ways to invoke a command procedure.)

Use the PROMPT and NOPROMPT operands on the EXEC and PROFILE commands to control prompting during the execution of command procedures. Note that if you specify NOPROMPT on PROFILE, no prompting is allowed even if you specify PROMPT on EXEC.

For a description of line continuation, see the section entitled "Line Continuation" in the introduction of this book.

Definitions of Command Procedure Terminology

Coding information for command procedures consists of built-in functions, control variables, operators, and statements, in addition to the set of commands and subcommands. Figure 12.1 is a reference list of this coding information. This lists, in alphabetic order, the character strings entered to control command procedures, an abbreviated meaning for each, and the heading in the book under which to find them.

Refer to **OS/VS2 Terminal User's Guide** for more how-to information on writing and using command procedures.

Expressions are part of command procedure terminology. Three types are defined.

- **Simple Expressions** consist of whole numbers, character strings, symbolic variables, system control variables and built-in functions; these are combined by arithmetic operators, parentheses, concatenation, and nesting of built-in functions.

Examples

```
3+4
( &A+&B/&C ) - ( 3*&D+4 )
&STR( ANS= )&SUBSTR( &EVAL( &N-2 ):&EVAL( &N+3 ),&BUF )
&VAL//2
```

- **Comparative Expressions** consist of simple expressions combined by comparison operators.

Examples

```
3+4=4+3
&N=&K
&X LT &Y
&A*2&B//3-5*( &C+6 ) <= &GOODVAL
&SUBSTR( 1:3, &ALPHABET )=&STR( ABC )
```

- **Logical Expressions** consist of comparative expressions combined by logical operators.

Examples

```
3+4=4+3 && &N >= &K
3+4 EQ 4+3 AND &N GE &K

&N >= &K | &X < &Y
&N GE &K OR &X LT &Y
```

- **Expressions** with no modifiers are assumed to be simple expressions.

See **OS/VS2 TSO Terminal User's Guide** for explanations of three topics that you should understand before using command procedure statements:

- Symbolic Variables
- Dynamic Symbol Substitution
- Concatenation

Name	Meaning	See
< (or LT)	Less than	Operators, Comparison
<= (or LE)	Less than or equal	Operators, Comparison
+	Addition	Operators, Arithmetic
| (or OR)	Or	Operators, Logical
&& (or AND)	And	Operators, Logical
&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

Figure 12.1. Command Procedure Coding Reference (Part 1 of 3)

Name	Meaning	See
*	Multiplication	Operators, Arithmetic
**	Exponentiation	Operators, Arithmetic
¬> (or NG)	Not greater than	Operators, Comparison
¬< (or NL)	Not less than	Operators, Comparison
¬= (or NE)	Not equal	Operators, Comparison
-	Subtraction	Operators, Arithmetic
/	Division	Operators, Arithmetic
//	Remainder	Operators, Arithmetic
> (or GT)	Greater than	Operators, Comparison
>= (or GE)	Greater than or equal	Operators, Comparison
= (or EQ)	Equal	Operators, Comparison
AND	And	Operators, Logical
ATTN	Attention exit	Statements
CLOSFILE	Close open file	Statements
CONTROL	Control options	Statements
DATA(-ENDDATA)	Starts DATA group	Statements
DO(-WHILE-END)	Start DO group	Statements
(IF-THEN-)ELSE	Start IF-not action	Statements
(DO-WHILE-)END	End DO group	Statements
END	End the command procedure	END Command
(DATA-)ENDDATA	Ends DATA group	Statements
EQ	Equal	Operators, Comparison
ERROR	Error exit	Statements
EXEC	Invoke a command procedure	EXEC Command
EXIT	Exit from nested procedure	Statements
GE	Greater than or equal	Operators, Comparison
GETFILE	Get record from open file	Statements
GLOBAL	Define global symbolic variables	Statements
GOTO	Unconditional branch	Statements
GT	Greater than	Operators, Comparison
IF(-THEN-ELSE)	Tests IF condition	Statements
LE	Less than or equal	Operators, Comparison
LT	Less than	Operators, Comparison
NE	Not equal	Operators, Comparison
NG	Not greater than	Operators, Comparison

Figure 12.1. Command Procedure Coding Reference (Part 2 of 3)

Name	Meaning	See
NL	Not less than	Operators,Comparison
OPENFILE	Open a file	Statements
OR	Or	Operators, Logical
PROC	Set and use symbolic parameters	Statements
PUTFILE	Put record into open file	Statements
READ	Get input from terminal	Statements
READDVAL	Get input from &SYSDVAL	Statements
RETURN	Return control from attn/err exit	Statements
SET	Assign values to variables	Statements
TERMIN	Request terminal input	Statements
(IF-)THEN(-ELSE)	Start IF action	Statements
WHEN	Inspect program return code	WHEN Command
(DO-)WHILE(-END)	DO loop control	Statements
WRITE	Send output to terminal	Statements
WRITENR	Send output to terminal with no return at end	Statements

Figure 12.1. Command Procedure Coding Reference (Part 3 of 3)

Operators

Command procedure operators specify operations to be performed on operands or on the terms in an expression. Operators are characterized as arithmetic, comparison, and logical.

Arithmetic

Arithmetic operators specify fixed-point arithmetic operations to be performed on numeric operands. The operands may be any simple expressions with numeric values.

The operator symbols may be either closed up to the associated operands or separated by one or more blanks.

For this function:	Enter:	Notes:
Addition	+	
Subtraction	-	
Multiplication	*	
Division	/	
Exponentiation	**	Negative exponents are assumed to be exponents of zero.
Remainder	//	

Comparison

Comparison operators specify compare operations to be done between two operands or expressions. The operands may be character or numeric values.

The comparison operators are specified either as characters or as symbols. When entered as characters, separate the operator from the operands with one or more blanks. When entered as symbols, the operator can be either contiguous with or set off by blanks from the operands.

For this function:	Enter symbols:	or characters:
Equal	=	EQ
Not Equal	¬=	NE
Less Than	<	LT
Greater Than	>	GT
Less Than or Equal	<=	LE
Greater Than or Equal	>=	GE
Not Greater Than	¬>	NG
Not Less Than	¬<	NL

Logical

Logical operators specify logical connections between expressions or conditions. The logical operators are specified either as characters or as symbols. If the character representation is used, the operators must be set off from their operands by blanks.

For this function:	Enter symbols:	or characters:
And	&&	AND
Or	|	OR

Evaluating Complex Expressions

In complex expressions, the order of associating operands with operators in the expression is determined by the priority of the operator. The operands connected by the highest priority operator (lowest number) are associated first. The operands connected by the second highest priority operator are associated next until all the considerations have been depleted.

If more than one operator of priority 1 appears in the expression, association of their operands is done from right to left. If more than one operator of the same priority, other than priority 1, appears in the expression, association of their operands is done from left to right.

If you want to change the order of association, you may use parentheses. Operators and operands enclosed in parentheses are associated first and are associated by priority.

For example:

```
SET &A=2*(3+4)
```

Note: (3+4) is associated first; the result is multiplied by 2.

```
SET &B=M*(N*(P/Q-R))
```

Note: (P/Q-R) is associated first; the result is multiplied by N; the result of multiplication is multiplied by M.

Operator	Operation	Priority	
+	Prefix plus	1	
-	Prefix minus	1	
**	Exponentiation	2	
//	Remainder	2	
*	Multiplication	3	
/	Division	3	Arithmetic expressions
+	Addition	4	
-	Subtraction	4	
=	Assignment	5	Comparative expressions
	Comparison Operators	5	
&&	AND	6	Logical expressions
|	OR	7	

Note: The above example shows the priority structure for the three different types of expressions.

Symbolic Substitution

Symbolic substitution causes real values (numeric or character data) to replace the symbolic variable names in the TSO commands, subcommands, and command procedure statements in a command procedure. Symbolic variables add flexibility to command procedures by symbolizing real values, which can change dynamically during command procedure execution. Because a variable may have different values at different times, command procedures using symbolic variables can execute multiple functions, or the same functions with multiple sets of values.

The term "symbolic variable" as used in this publication is a general one that refers to any command procedure character string for which differing values may be substituted at different times. This publication also refers to other specific kinds of symbolic variables:

- Symbolic parameters
- System control variables
- Global variables
- File names on file I/O statements

Parameters are the special variable names on PROC, READ, and READDVAL statements. They constitute a subset among symbolic variables because they represent values that pass from the user to the command procedure.

Rules for Symbolic Variables

The rules for symbolic variables apply uniformly to all symbolic representation in command procedures, whether they are parameters, control variables, or built-in functions.

- Symbolic variables consist of an ampersand symbol (&) and 31 or fewer alphameric characters, the first of which is alphabetic.
- The leading ampersand is mandatory, with the following exceptions:

STATEMENT	USE OF AMPERSAND
SET	Optional on left side of equal sign
READ/READDVAL	Optional
GLOBAL	Optional
PROC	Not permissible
File I/O Statements	Not permissible for evaluated filename

- **Establishing symbolic variables** consists of including their previously unused symbolic variable names on SET, GLOBAL, READ, READDVAL, PROC, or OPENFILE statements. Symbolic substitution and concatenation can be used to establish variables on SET and OPENFILE statements.
- **Substitution of symbolic variables** occurs line by line. Substitution is the process of replacing the variable names with real values.
- **Line scanning** progresses from left to right. For lines with unnested variables, substitution occurs in line scanning order.
- **Nested symbolic variables** are those for which the symbolic substitution routine cannot make all substitutions in a single scan because the values substituted for some variables are in turn symbolic variables.
- **The line scanning limit** is a value in the &SYSSCAN control variable. It limits the number of times symbolic substitution may scan a line to substitute values for all symbolic variables.
- **Concatenation of symbolic variables** consists of writing their names one after another with their ampersands, but with no intervening delimiters, for example:

  ```
  &A&B&C
  ```

- **Substitution of concatenated variables** is from left to right, in normal line scanning order.
- **Concatenating variables to character strings** consists of following the variable name immediately with a period and the string, for example:

  ```
  &VARIABLE.ALPHA
  ```

- **Substitution for double ampersands** results in a single ampersand for each pair before execution. This substitution takes place only after all other substitution in a line is complete, provided the scanning limit permits complete substitution. Substitution for double ampersands results regardless of the scanning limit, because the symbolic substitution routine groups pairs of ampersands for later substitution independent of any scanning for the substitution of values for variables. Therefore, symbolic substitution regards the expression "&&X" as a string consisting of a pair of ampersands followed by the characters "X", and "&&&X" as a string consisting of a pair of ampersands followed by the symbolic variable name "&X". For example, consider the following statement:

  ```
  SET &X  =   &STR(&&X)
  ```

Note: One exception to the general rule of substitution for the double ampersand is the file variable, where the double ampersands will remain.

After symbolic substitution, the value of &X is the string "&X", which is a symbolic variable name in terms of command procedure execution.

Note: A single ampersand, by itself, is an invalid command procedure expression, which will cause an error. See the sections "Command Procedure Operators" and "The AND Operator" in the **OS/VS2 Terminal Users Guide.**

Symbolic Substitution Examples

The following examples illustrate the major rules for symbolic substitution.

Example 1

This example illustrates the substitution of the innermost built-in functions prior to substitution of the outer levels in an expression with nested symbolic variables. In the following expression, assume these initial values for the symbolic variables:

```
&START = 2; &FINISH = 3; &STRING = ABCDE
```

Then consider the following lines as the sequence through which symbolic substitution substitutes values for all the variables:

```
Original expression:&LENGTH(&SUBSTR(&START:&FINISH,&STRING))
Intermediate result:&LENGTH(&SUBSTR(2:3,ABCDE))
Result of scan 1: &LENGTH(BC)
Result of scan 2: 2
```

Complete substitution in this example takes only two scans. The "intermediate result" shown is for clarifying the substitution part of the process that occurs during the first scan of the line. The complete resolution of scan 1 also includes the evaluation of the built-in function "&SUBSTR", which by definition is an immediate evaluation. Therefore, the result of both the symbolic substitution of values for variables and the evaluation of the built-in function appear in the following line, as the "result of scan 1." Scan 2 then resolves the built-in function "&LENGTH" to produce the final result of "2".

Example 2

This example illustrates concatenation. Assume that a procedure uses the following three assignment statements:

```
SET &P1 = CONCA
SET &P2 = ATION
SET &A = VAR
```

Then the substitution will take place in the following statement as shown:

```
Original statement:       SET &NEW&A.1 = &P1.TEN&P2
End of substitution:      SET &NEWVAR1 = CONCATENATION
```

Note that a period is necessary to concatenate a character string to a symbolic variable. Omission of the period would cause TSO to try to find a value for the variable &A1. An error would result from this notation if the procedure did not elsewhere establish a value for the variable &A1.

Example 3

Example 3 illustrates how to use concatenation and symbolic substitution on the left side of the SET statement to create subscripted variable names:

```
set &x = 1                 /* initialize x                  */
do while &x le 2           /* x-dimension is 2              */
    set &y = 1             /* initialize y                  */
    do while &y le 3       /* y-dimension is 3              */
        set x&x.y&y = 0 /* create 2-by-3 array of zeros */
        set &y = &y + 1 /* increment the y-dimension     */
    end                    /*                               */
    set &x = &x + 1        /* increment the x-dimension     */
end                        /*                               */
```

This command procedure creates an array consisting of the following six symbolic variables:

```
&X1Y1   &X1Y2   &X1Y3
&X2Y1   &X2Y2   &X2Y3
```

Built-In Functions

Built-in functions perform evaluations of data expressions, character strings, and substrings. To use a built in function, include the appropriate symbolic variable on a command procedure statement. TSO substitutes data for the symbolic variable at execution time.

&DATATYPE

&DATATYPE(expression)

Use &DATATYPE to find out whether an expression is entirely numeric. TSO replaces the variable with either NUM for all numeric or CHAR for anything else.

&EVAL

&EVAL(expression)

Use &EVAL to find the result of an arithmetic expression. TSO replaces the variable with the calculated numeric value.

&LENGTH

&LENGTH(expression)

Use &LENGTH to find the number of characters in a result calculated for an expression. The numeric value that TSO substitutes for the variable pertains to the calculated result, not to the number of characters in the expression itself.

&STR

&STR(string)

Use &STR to enter a specific value in place of the variable. This function suppresses the evaluation of the string but does not suppress symbolic substitution or the evaluation of built-in functions within the string.

&SUBSTR

&SUBSTR(expression[:expression],character-string)

Use &SUBSTR to select a range from a character string and put that range, or substring, in place of the variable. The start and end of the substring are specified numerically as the nth positions of the character string. The range can be from one to all the characters in the string. The numeric values can be directly entered or can be the calculated results of symbolic expressions.

To select a one-character substring, you may enter the start-of-substring parameter and the character string. The system assumes the end-of-substring value to be the start-of-substring value.

Control Variables

Control variables contain character-format information about the current command procedure environment and user. This information can be used by command procedures to establish, for example, conditions for branching.

Four of the control variables can be set by command procedures. The remaining control variables are maintained by the system; attempts to modify them produce errors. The variables you can set are &LASTCC, &MAXCC, &SYSDVAL, and &SYSSCAN.

&LASTCC

&LASTCC

Use &LASTCC to get the return code form the last operation, whether TSO command/subcommand or statement. The system return codes are in **OS/VS Message Library: VS2 System Codes.** The command procedure return codes are in Figure 12.2. &LASTCC is modifiable.

&MAXCC

&MAXCC

Use &MAXCC to find the highest return code issued up to now in the procedure, or passed back from any nested procedures that are executed. &MAXCC is modifiable. The return code will be returned in decimal format.

&SYSDATE

&SYSDATE

Use &SYSDATE to get the present date in the format mm/dd/yy where mm is month, dd is day of month, and yy is year. &SYSDATE is not modifiable.

&SYSDLM

&SYSDLM

Use &SYSDLM to identify which delimiter string of those specified by the TERMIN statement the user entered to give up control. This allows a user to select options that a command procedure may provide. &SYSDLM is not modifiable.

&SYSDVAL

&SYSDVAL

Use &SYSDVAL for either of two purposes:

- When TERMIN passes control to the terminal user, get the value of any parameters the user enters besides the delimiter string entered to pass control back to the procedure.
- When READ requests specific terminal input, get the value of the user response line. &SYSDVAL is modifiable.

&SYSICMD

&SYSICMD

Use &SYSICMD to get the name by which the user implicitly invoked the currently executing command procedure. The name value will be null if the user invoked this command procedure explicitly. &SYSICMD is not modifiable.

&SYSNEST

&SYSNEST

Use &SYSNEST to get a character string YES or NO, signifying whether the currently executing command procedure is nested (invoked from another command procedure instead of from the terminal directly). &SYSNEST is not modifiable.

&SYSPCMD

&SYSPCMD

Use &SYSPCMD to get the name of the most recently executed TSO command in this procedure. The initial value is 'EXEC' in the command environment and 'EDIT' in the subcommand environment. &SYSPCMD is not modifiable.

&SYSPREF

&SYSPREF

Use &SYSPREF to get the data-set-name prefix specified in the user profile table of the command procedure user. The user can set &SYSPREF by issuing the PROFILE command but cannot modify it from the command procedure.

&SYSPROC

&SYSPROC

Use &SYSPROC to get the logon procedure name for the current command procedure user. &SYSPROC is not modifiable.

&SYSSCAN

&SYSSCAN

Use &SYSSCAN to get the maximum value for the number of times that symbolic substitution is allowed to rescan a line to resolve symbolic variables. &SYSSCAN is modifiable; default value is 16, minimum value is 0, maximum is largest fixed-point number.

&SYSSCMD

&SYSSCMD

Use &SYSSCMD to get the name of the subcommand currently executing. The initial value is null if EXEC is issued in the command environment and 'EXEC' when issued as a subcommand of EDIT. The value is null whenever the procedure is in the command environment. &SYSSCMD is not modifiable.

&SYSTIME

&SYSTIME

Use &SYSTIME to get the present time in the format hh:mm:ss where hh is hours, mm is minutes, and ss is seconds. &SYSTIME is not modifiable.

&SYSUID

&SYSUID

Use &SYSUID to get the user identification (userid) of the individual currently executing the command procedure. &SYSUID is not modifiable.

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 12.3.

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 12.2. Command Procedure Statement Categories

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.

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. This action can effectively be an attention exit.

```
[label:]          ATTN  [OFF   ]
                        [action]
```

label: specifies a name to which the command procedure can branch. Enter 1-8 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. 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 1-8 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).

CONTROL Statement

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 1-8 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 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 1-8 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 1-8 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 inptu 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.

```
[label:] ERROR  [OFF   ]
                [action]
```

label: specifies a name to which the command procedure can branch. Enter 1-8 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 1-8 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 1-8 chracters, 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).

GLOBAL Statement

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 1-8 characters, the first alphabetic and the rest alphameric, followed by a colon and at least one blank.

name1-naneN specfy 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.

GOTO Statement

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 1-8 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 statement extends to the next line.

```
[label:]          IF logical-expression THEN [action]
                  [ELSE [action]]
```

label: specifies a name to which the command procedure can branch. Enter 1-8 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 true. The ELSE action is invoked if the IF condition is false and ELSE is specified. If the IF condition is false 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.

Note: 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.

Note: 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/VS Data Management Services Guide.**

```
[label:]          OPENFILE filename  [INPUT ]
                                     [OUTPUT]
                                     [UPDATE]
```

label: specifies a name to which the command procedure can branch. Enter 1-8 characters, the first alphabetic and the rest alphameric, followed by a colon and at least one blank.

filename specifies the name (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 no types are entered.

OUTPUT specifies that the file will open for output.

UPDATE specifies that the file will open for update.

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 1-8 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 */
.
.
```

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 1-8 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. 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.

Syntactical words are defined as character strings, quoted strings, parenthesized strings, or null values as two adjacent commas (,,) or quotes (' ').

[label:]	**READDVAL**	[name1 [name2 . . . . nameN]]

label: specifies a name to which the command procedure can branch. Enter 1-8 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 1-8 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**	$\left\{ \begin{array}{l} = \\ \text{EQ} \end{array} \right\}$ **expression**

label: specifies a name to which the command procedure can branch. Enter 1-8 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 (with &) or allowable control variable (with &) 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.

[label:]	**TERMIN**	[string1 / ,] [string2 . . . . stringN]

label: specifies a name to which the command procedure can branch. Enter 1-8 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 1-8 characters, the first alphabetic and the rest alphameric, followed by a colon and at least one blank.

WRITE statement specifies that the output at the terminal has a carrier return at the end of the line.

WRITENR statement specifies that the carrier does not return after the text is printed.

text specifies what is to be sent to the terminal. You can enter any character string, including symbolic variables.

Example

Operation: Illustrates WRITE and WRITENR usage.

```
.
.
WRITENR ONE
WRITENR TWO/
WRITENR THREE
WRITE FOUR/*Terminal output, first line:ONETWO/THREEFOUR*/
WRITE FIVE /* Terminal output next line:   FIVE */
.
.
```

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	(Reserved)
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
999	Internal command procedure error
* Sxxx	A system ABEND code
* Uxxx	A user ABEND code

* Printed in hexadecimal

Figure 12.3. Command Procedure Statement Error Codes (Decimal)

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 a single character 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 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 for 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 13 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 CHARACTER+ -
    a
    JOB SMITH3A(JOB00071) SUBMITTED
    READY
```

Figure 13. 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 13, 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 either:

- Explicitly via HOLD=YES on a DD statement or via the ALLOCATE or FREE command, or

Example JCL for an explicitly held data set

```
//SMITH6     JOB      MSGLEVEL=1,MSGCLASS=C,NOTIFY=SMITH
//           EXEC         PGM=IEBDG
//SYSPRINT   DD       SYSOUT=M,HOLD=YES
//
//  remainder of JCL statements
//
```

- Implicitly by specifying an installation-defined reserved class for SYSOUT and MSGCLASS. It is not necessary to have them reserved in the same class.

```
//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 and 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 listed 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 listed 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 RETURN 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 completes and you used the PAUSE operand in the OUTPUT command.
- You press the attention key.

Note: An attention interruption can cause unpredictable results in the print processing. When attention 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 printing 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 listed at your terminal. The system pauses after printing 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 printing at the beginning of the data set being processed. To resume printing at the beginning, enter:

```
OUTPUT
continue begin
```

If you prefer to resume printing 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 list 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 cancelled successfully. A message will also be displayed at the system operator's console when a job is cancelled.

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.

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

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 cancelled 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.

```
{OUTPUT}  (jobname[(jobid)]-list)
{OUT   }  [CLASS(classname-list)]
          [[PRINT[({*      })]] [BEGIN] [PAUSE  ] [KEEP[HOLD  ]]
          [      [ ({dsname})]] [HERE ] [NOPAUSE] [    [NOHOLD]]
          [                     [NEXT ]           [NOKEEP      ]
          [DELETE]
          [NEWCLASS(classname)]  [DEST(remote-station-id)] [HOLD  ]
          [                                                [NOHOLD]]
```

(job-name[(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(class-name-list) specifies the names of the output classes to be searched for output from the jobs identified in the jobname list. If you do not specify the name of 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(data-set-name 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.

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. (A carrier return entered after the pause will cause normal processing to continue.) This operand can be overridden by the NOPAUSE operand of the CONTINUE subcommand.

NOPAUSE indicates that output operations are not to be interrupted. This operand can be overridden by the PAUSE operand of the CONTINUE subcommand. 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.

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 listed 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.

```
{CONTINUE}   [BEGIN]
{C       }   [HERE ]
             [NEXT ]
             [PAUSE  ]
             [NOPAUSE]
```

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 attention 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. (A carrier return entered after the pause will cause normal processing to continue.) You can use this operand to override a previous NOPAUSE condition for output.

NOPAUSE indicates that output operations are not to be interrupted. You can use this operand to override a previous 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}        data-set-name
{S   }
```

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 this character. SUBMIT 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.

$$\left\{\begin{array}{l}\textbf{SUBMIT}\\\textbf{SUB}\end{array}\right\}\quad\textbf{(data-set-list)}\quad\left[\begin{array}{l}\underline{\text{NOTIFY}}\\\text{NONOTIFY}\end{array}\right]$$

(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 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 command may be used only by personnel who have been given the authority to do so by the installation management.
- SUBMIT does not support job entry subsystem control cards which precede the JOB card.

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.

Appendix B: Program Product Commands

ASM Command

The ASM command is provided as part of the optional TSO ASM Prompter program product, which is available for a license fee.

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.

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.

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 14.

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 14. 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.

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.

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.

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.

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.

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 TESTFORT command).

GOFORT Command

The GOFORT command is provided as part of the optional TSO Code and Go FORTRAN processor. 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 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.

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.

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. 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. 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. Used in conjunction with Full American National Standard COBOL Version 4, COBOL Interactive conjunction with Code and Go FORTRAN or FORTRAN IV (G1), FORTRAN Interactive Debug provides comprehensive capabilities for program monitoring and checkout.

TESTFORT Command

The TESTFORT command is provided as part of the optional FORTRAN Interactive Debug program product, which is available for a license fee. 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:

ALTER 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.

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**.

Index

OS/VS2 TSO Command Language Reference

GC28-0646-3

READER'S COMMENT FORM

Your views about this publication may help improve its usefulness; this form will be sent to the author's department for appropriate action. Using this form to request system assistance or additional publications will delay response, however. *For more direct handling of such requests, please contact your IBM representative or the IBM Branch Office serving your locality.*

Possible topics for comment are:

Clarity Accuracy Completeness Organization Index Figures Examples Legibility

What is your occupation? ______________________________

Number of latest Technical Newsletter (if any) concerning this publication: ______________

Please indicate your address in the space below if you wish a reply.

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments.)

Cut or Fold Along Line

Your comments, please . . .

This manual is part of a library that serves as a reference source for system analysts, programmers, and operators of IBM systems. Your comments on the other side of this form will be carefully reviewed by the persons responsible for writing and publishing this material. All comments and suggestions become the property of IBM.

Cut or Fold Along Line

Fold

Fold

First Class
Permit 81
Poughkeepsie
New York

Business Reply Mail

No postage stamp necessary if mailed in the U.S.A.

Postage will be paid by:

International Business Machines Corporation
Department D58, Building 706-2
PO Box 390
Poughkeepsie, New York 12602

Fold

Fold

IBM

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)

OS/VS2 TSO Command Language Reference (S370-39) Printed in U.S.A. GC28-0646-3

IBM®

International Business Machines Corporation
Data Processing Division
1133 Westchester Avenue, White Plains, New York 10604
(U.S.A. only)

IBM World Trade Corporation
821 United Nations Plaza, New York, New York 10017
(International)