z/OS
Version 2 Release 3

TSO/E Command Reference

SA32-0975-30

Note

Before using this information and the product it supports, read the information in "Notices" on page 399.

This edition applies to Version 2 Release 3 of z/OS (5650-ZOS) and to all subsequent releases and modifications until otherwise indicated in new editions.

Last updated: 2019-02-16

Contents

List of Figures

List of Tables

About this document

This document supports z/OS (5650-ZOS).

This document describes the syntax and function of the commands and subcommands of the TSO/E command language and Session Manager. It provides only reference material and assumes you are experienced in the use of TSO/E and Session Manager.

If you are not familiar with TSO/E, first read *z/OS TSO/E User's Guide*. If you have little or no knowledge of the use of TSO/E commands, *z/OS TSO/E User's Guide*, provides the prerequisite information for using this document. The guide explains how to:

- Enter and execute commands
- Name and create specific types of data sets
- Edit, rename, list, copy, free, and delete data sets
- Send and receive data sets
- Print data sets on a JES printer
- Run programs in the foreground or background
- Use TSO/E through ISPF/PDF
- Use Session Manager

Note:

1. System programming commands are described in *z/OS TSO/E System Programming Command Reference*.
2. When you see the term JESPLEX in this publication, understand it to mean either a logical grouping of JES2 systems that share the same multi-access spool (MAS) or a logical grouping of JES3 systems (each JES3 system consisting of one global JES3 system and some number of local JES3 systems).

Who should use this document

Anyone who uses TSO/E and Session Manager commands.

How this document is organized

The major chapters in this document are:

- Chapter 1, "TSO/E commands and subcommands," on page 1 contains general information needed to use TSO/E commands. It describes the syntax notation in diagrams that accompany each command, positional and keyword operands, delimiters, line continuation, comments, and subcommands.

  This document presents commands in alphabetical order. The subcommands are alphabetized under their commands. For example, all TEST subcommands are alphabetized under the TEST command. Examples are included.
- Chapter 2, "Session Manager commands," on page 361 describes the syntax and function of each Session Manager command. It presents the commands in alphabetical order and includes examples.

Where to find more information

See *z/OS Information Roadmap* for an overview of the documentation associated with z/OS®, including the documentation available for z/OS TSO/E.

Introductory information about how to use TSO/E is described in *z/OS TSO/E User's Guide*.

How to read syntax diagrams

This section describes how to read syntax diagrams. It defines syntax diagram symbols, items that may be contained within the diagrams (keywords, variables, delimiters, operators, fragment references, operands) and provides syntax examples that contain these items.

Syntax diagrams pictorially display the order and parts (options and arguments) that comprise a command statement. They are read from left to right and from top to bottom, following the main path of the horizontal line.

For users accessing the Information Center using a screen reader, syntax diagrams are provided in dotted decimal format.

Symbols

The following symbols may be displayed in syntax diagrams:

Symbol
Definition

►►—
Indicates the beginning of the syntax diagram.

—►
Indicates that the syntax diagram is continued to the next line.

►—
Indicates that the syntax is continued from the previous line.

—►◄
Indicates the end of the syntax diagram.

Syntax items

Syntax diagrams contain many different items. Syntax items include:

- Keywords - a command name or any other literal information.
- Variables - variables are italicized, appear in lowercase, and represent the name of values you can supply.
- Delimiters - delimiters indicate the start or end of keywords, variables, or operators. For example, a left parenthesis is a delimiter.
- Operators - operators include add (+), subtract (-), multiply (*), divide (/), equal (=), and other mathematical operations that may need to be performed.
- Fragment references - a part of a syntax diagram, separated from the diagram to show greater detail.
- Separators - a separator separates keywords, variables or operators. For example, a comma (,) is a separator.

Note: If a syntax diagram shows a character that is not alphanumeric (for example, parentheses, periods, commas, equal signs, a blank space), enter the character as part of the syntax.

Keywords, variables, and operators may be displayed as required, optional, or default. Fragments, separators, and delimiters may be displayed as required or optional.

Item type
Definition

Required
Required items are displayed on the main path of the horizontal line.

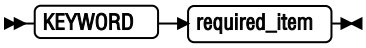
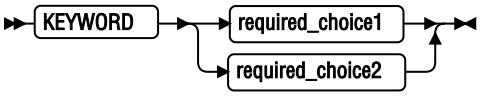
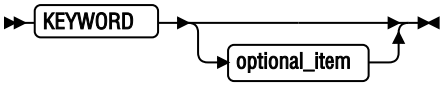
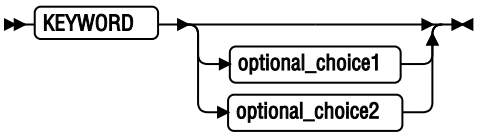
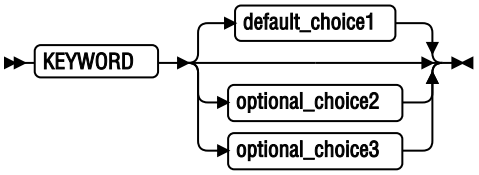
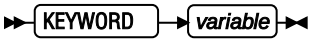
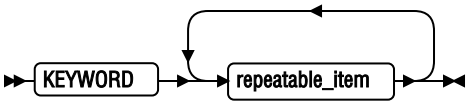
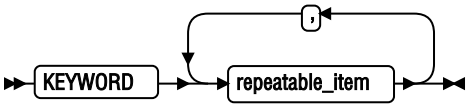
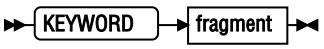
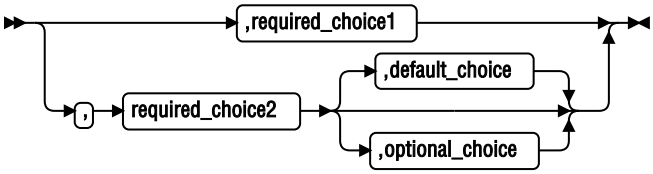
Optional
Optional items are displayed below the main path of the horizontal line.

Default
Default items are displayed above the main path of the horizontal line.

Syntax examples

The following table provides syntax examples.

Table 1: Syntax examples

Item	Syntax example
Required item. Required items appear on the main path of the horizontal line. You must specify these items.	
Required choice. A required choice (two or more items) appears in a vertical stack on the main path of the horizontal line. You must choose one of the items in the stack.	
Optional item. Optional items appear below the main path of the horizontal line.	
Optional choice. An optional choice (two or more items) appears in a vertical stack below the main path of the horizontal line. You may choose one of the items in the stack.	
Default. Default items appear above the main path of the horizontal line. The remaining items (required or optional) appear on (required) or below (optional) the main path of the horizontal line. The following example displays a default with optional items.	
Variable. Variables appear in lowercase italics. They represent names or values.	
Repeatable item. An arrow returning to the left above the main path of the horizontal line indicates an item that can be repeated. A character within the arrow means you must separate repeated items with that character. An arrow returning to the left above a group of repeatable items indicates that one of the items can be selected,or a single item can be repeated.	 
Fragment. The fragment symbol indicates that a labelled group is described below the main syntax diagram. Syntax is occasionally broken into fragments if the inclusion of the fragment would overly complicate the main syntax diagram.	 fragment 

How to send your comments to IBM

We invite you to submit comments about the z/OS product documentation. Your valuable feedback helps to ensure accurate and high-quality information.

Important: If your comment regards a technical question or problem, see instead "If you have a technical problem" on page xxi.

Submit your feedback by using the appropriate method for your type of comment or question:

Feedback on z/OS function
If your comment or question is about z/OS itself, submit a request through the IBM RFE Community (www.ibm.com/developerworks/rfe/).

Feedback on IBM® Knowledge Center function
If your comment or question is about the IBM Knowledge Center functionality, for example search capabilities or how to arrange the browser view, send a detailed email to IBM Knowledge Center Support at ibmkc@us.ibm.com.

Feedback on the z/OS product documentation and content
If your comment is about the information that is provided in the z/OS product documentation library, send a detailed email to mhvrcfs@us.ibm.com. We welcome any feedback that you have, including comments on the clarity, accuracy, or completeness of the information.

To help us better process your submission, include the following information:

- Your name, company/university/institution name, and email address
- The following deliverable title and order number: z/OS TSO/E Command Reference, SA32-0975-30
- The section title of the specific information to which your comment relates
- The text of your comment.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute the comments in any way appropriate without incurring any obligation to you.

IBM or any other organizations use the personal information that you supply to contact you only about the issues that you submit.

If you have a technical problem

If you have a technical problem or question, do not use the feedback methods that are provided for sending documentation comments. Instead, take one or more of the following actions:

- Go to the IBM Support Portal (support.ibm.com).
- Contact your IBM service representative.
- Call IBM technical support.

Summary of changes

This information includes terminology, maintenance, and editorial changes. Technical changes or additions to the text and illustrations for the current edition are indicated by a vertical line to the left of the change.

Summary of changes for z/OS Version 2 Release 3 (V2R3)

The following changes are made for z/OS Version 2 Release 3 (V2R3).

New

- The limit for TSO/E user IDs is changed to 8 characters. For more information, see:
  - "CANCEL command operands" on page 66
  - "LISTBC command" on page 165
  - "LOGON command" on page 182
  - "OUTDES command" on page 188
  - "PRINTDS command" on page 215
  - "PROFILE command operands" on page 229
  - "PROFILE foreground/background processing differences" on page 232
  - "RECEIVE command" on page 239
  - "SEND command operands" on page 255
  - "STATUS command operand" on page 265
  - "SUBMIT command examples" on page 269, "Example 2" on page 269
  - "TRANSMIT command" on page 334
- Modified "LISTALC command" on page 162 to clarify allocation of data sets.
- Modified "EDIT command" on page 70 to add a reference on more information about line mode edit.
- Modified "LOGON command syntax" on page 184 and "LOGON command operands" on page 184 adding the SUBSYSTEM operand.
- Modified "SUBMIT command syntax" on page 266 and "SUBMIT command operands" on page 267 adding the SUBSYSTEM operand.

Changed

- "ALLOCATE command operands" on page 16 is updated to clarify referencing generation data sets.
- With APAR OA53331, "TEST command" on page 274 is updated to include the RMODE 64 module in restrictions.

Summary of changes for z/OS Version 2 Release 2 (V2R2)

The following changes are made for z/OS Version 2 Release 2 (V2R2).

New

No content was added.

Changed

- With APAR OA43898, modified ALLOCATE command by adding MAXGENS operand to syntax and operands. For details, see the topics about “ALLOCATE command syntax” on page 11 and “ALLOCATE command operands” on page 16.
- Modified LINK command by adding INFORM option to LISTPRIV operand. For details, see the topic about “LINK command syntax” on page 149 and “LINK command operands” on page 152.
- Modified LOADGO command by adding INFORM option to LISTPRIV operand. For details, see the topics about “LOADGO command syntax” on page 174 and “LOADGO command operands” on page 175.
- Modified “Full-Screen LOGON versus line mode LOGON” on page 182 of LOGON command.
- Modified “Full-Screen LOGON processing” on page 183 of LOGON command.

Deleted

- With APAR OA47704, the note LISTDS command with the STATUS option returns blanks for all but the first DDNAME when issued for concatenated DDNAMES is removed from the STATUS operand of the LISTDS command.

z/OS Version 2 Release 1 summary of changes

See the Version 2 Release 1 (V2R1) versions of the following publications for all enhancements related to z/OS V2R1:

- *z/OS Migration*
- *z/OS Planning for Installation*
- *z/OS Summary of Message and Interface Changes*
- *z/OS Introduction and Release Guide*

Chapter 1. TSO/E commands and subcommands

This chapter describes the functions and syntax of TSO/E commands and their subcommands. It includes:

- The general format and syntax rules for the commands
- A description of each command. The commands are described in alphabetical order.
- Examples of how to use commands and subcommands.

The commands are presented in alphabetical order. Subcommands are also presented in alphabetical order following the command to which they apply.

Introductory information about how to use TSO/E is described in *z/OS TSO/E User's Guide*.

Using a TSO/E command

A command consists of a command name typically followed by one or more operands. Operands provide the specific information required to perform the requested operation. For example, operands for the RENAME command identify the data set you want to rename:

RENAME	OLDNAME	NEWNAME
command name	operand_1 (old data set name)	operand_2 (new data set name)

You can use two types of operands with the commands: *positional* and *keyword*.

Positional operands

Positional operands follow the command name in a certain order. In the command descriptions within this book, the positional operands are shown in lowercase characters. For example,

```
EDIT reports.data
```

where *reports.data* is the *data_set_name* positional operand with the EDIT command.

When you enter a positional operand that is a list of several names or values, you must enclose the list within parentheses. For example,

```
LISTDS (PARTS.DATA TEST.DATA)
```

Keyword operands

Keyword operands (keywords) are specific names or symbols that have a particular meaning to the system. You can include keywords in any order following the positional operands. In the command descriptions within this book, keywords are shown in uppercase characters.

You can specify values with some keywords. Enclose the value with parentheses following the keyword. For example, a typical keyword operand with a value is:

```
LINESIZE(integer)
```

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

```
LINESIZE(80)
```

However, if you enter conflicting, mutually exclusive keywords, the last keyword you enter overrides the previous ones.

"How to read the TSO/E command syntax" on page 2 describes the syntax notation for the TSO/E commands and subcommands.

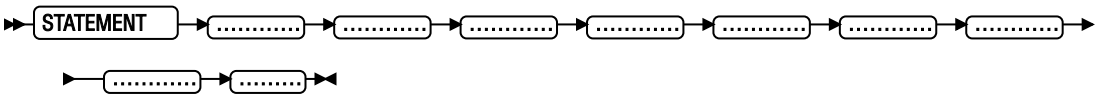
How to read the TSO/E command syntax

Throughout this information, syntax of the whole command is described using the structure defined later in this topic.

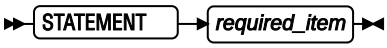
Read the syntax diagrams from left to right, from top to bottom, following the path of the line.

Double arrows indicate the beginning and ending of a statement.

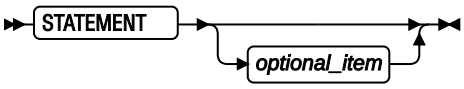
If a statement syntax requires more than one line to be shown, single arrows indicate their continuation.



Required items appear on the horizontal line (the main path).

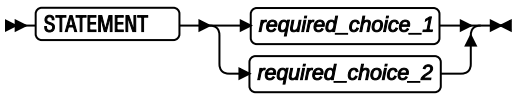


Optional items appear below the main path.

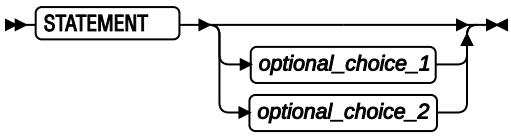


If you can choose from two or more items, they are stacked vertically.

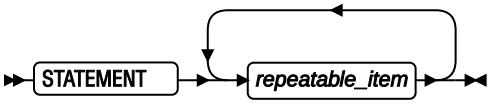
- If you *must* choose one of the items, an item of the stack appears on the main path.



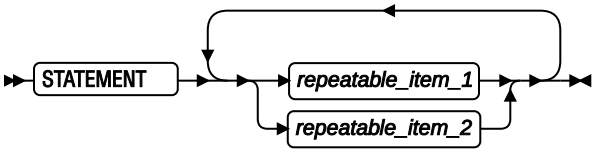
- If choosing one of the items is optional, the entire stack appears below the main path.



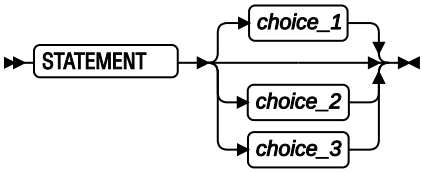
An arrow returning to the left above the main line indicates an item that can be repeated.



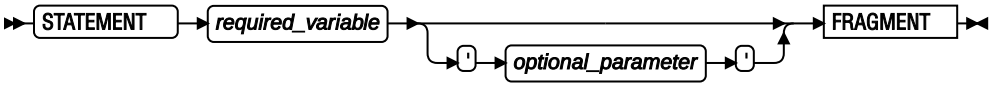
A repeat arrow above a stack indicates that you can make more than one choice from the stacked items, or repeat a single choice.



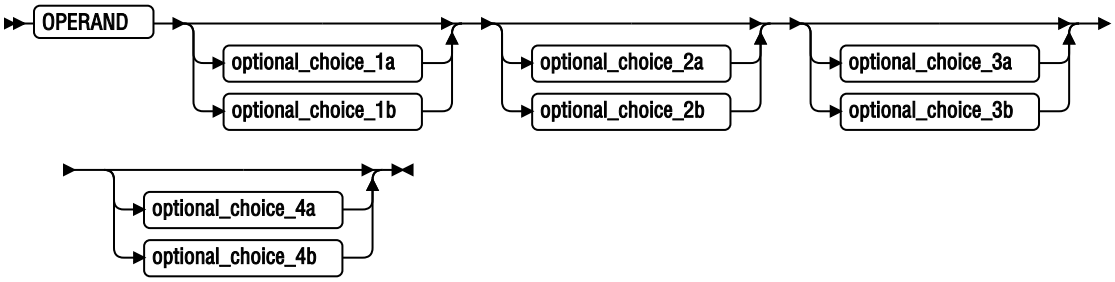
Default values appear above the main path. For example, if you choose neither *choice_2* nor *choice_3*, *choice_1* is assumed. (Defaults *can* be coded for clarity reasons.)



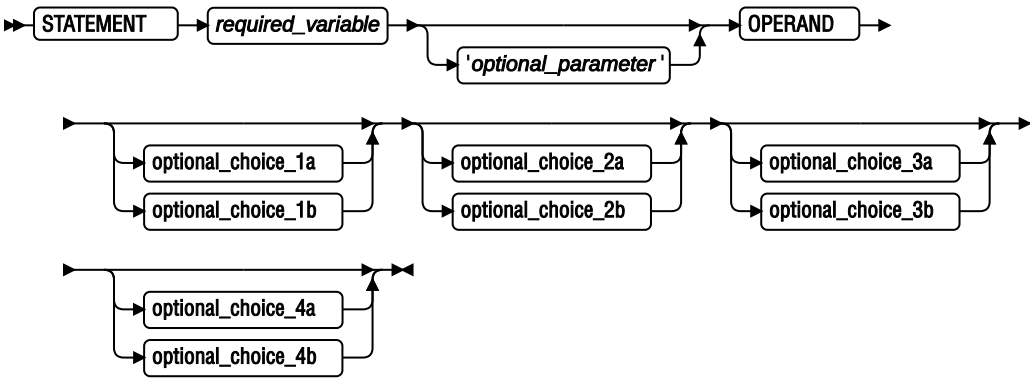
If a syntax diagram becomes too large or too complex to be printed or shown, fragments of it are shown below the main diagram as details.



FRAGMENT



The previous syntax diagram is equivalent to the following diagram:



Keywords appear in uppercase (for example, PARM1). They can be spelled exactly as shown, but they can be in mixed or lower case. Variables appear in all lowercase letters (for example, *parmx*). They represent user-supplied names or values.

If punctuation marks, parentheses, arithmetic operators, or such symbols are shown, they must be entered as part of the syntax.

Abbreviating keyword operands

You can enter keywords spelled exactly as they are shown or you can use an acceptable abbreviation. You can 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 example, the LISTBC command has four keywords:

- MAIL
- NOMAIL
- NOTICES
- 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)

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

Comments

You can include comments in a TSO/E command anywhere a blank might appear. To include a comment, start with delimiter /*. If you want to continue the command after the comment, close the comment with delimiter */.

```
listd (data_set_list) /* my data sets
```

or

```
listd /* my data sets */ (data_set_list)
```

You do not need to end a comment with */ if the comment is the last thing on the line. Ending a comment with */ is a convention, not a requirement here. Comments are most useful in CLISTs.

Line continuation

CAUTION: A plus sign causes leading delimiters to be removed from the continuation line.

When it is necessary to continue to the next line, use a plus (+) or minus (-) sign as the last character of the line you want to continue.

```
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
```

Note: If you are using REXX commands and want to continue to the next line, the plus or minus sign does not work. You must use the comma.

The following example shows how to use the comma with the REXX command (PUSH), to continue to the next line. The comma must be outside the quotation marks.

```
/* REXX * test ACCOUNT */
x = Outtrap("var.")
PUSH 'END'
PUSH 'ADD  (NEWUSER * * TPROC)',
```

```
     'UNIT(SYSTS)',
     'SIZE(4000)'
Address TSO "ACCOUNT"
x = Outtrap("OFF")
Say 'RC from account was:' rc
Do i=1 to var.0               /* loop through all messages */
  Say var.i                   /* display each message      */
End
```

To continue a line that contains a comment, use a continuation character *after* the comment:

```
allocate dataset(my.text) /* data set name */ +
         new volume(tsomar2)
```

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 any character you enter after a semicolon is ignored. For example, if you use a blank or a comma as a delimiter, you can type the LISTBC command as follows:

```
LISTBC NOMAIL NONOTICES
LISTBC NOMAIL,NONOTICES
LISTBC NOMAIL   NONOTICES
```

When creating (or updating) a CLIST, do not use any of the following as a delimiter:

- The special characters @, $, or #
- A single quotation mark
- A number
- A blank
- A tab
- A comma
- A semicolon
- A parenthesis
- An asterisk

Note: When entering commands under ISPF or Program Control Facility (PCF), do not use the ISPF or PCF command delimiter character that your installation has set for these facilities. The default delimiter character for each ISPF and PCF command is the semicolon (;), but your installation can specify a different delimiter character.

Using the HELP command

Use the HELP command to receive all the information about the system on how to use any TSO/E command. The requested information is displayed on your terminal.

Explanations of commands

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

```
help
```

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

```
help ALLOCATE
```

where ALLOCATE is the command name.

Syntax interpretation of HELP information

The syntax notation for the HELP information is different from the syntax notation presented in this book because it is restricted to characters that are displayed on your terminal. You can get the syntax interpretation by entering the HELP command as follows:

```
help help
```

Explanations of subcommands

When HELP exists as a subcommand, you can 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 commands for VSAM and Non-VSAM data sets

Access Method Services is a multi-function service program that primarily establishes and maintains Virtual Storage Access Method (VSAM) data sets.

Table 2 on page 6 shows recommended commands, by function, for VSAM and non-VSAM data sets. Numbers indicate order of preference. Licensed program commands are identified with an asterisk (*). For commands not covered in this book, see *z/OS DFSMS Access Method Services Commands*.

Table 2: Commands preferred for VSAM/Non-VSAM data sets

Function	Non-VSAM	VSAM
Build lists of attributes	ATTRIB	(None)
Allocate new DASD space	ALLOCATE	DEFINE or ALLOCATE
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	1. LISTDS, 2. LISTCAT	1. LISTCAT, 2. LISTDS
List names of cataloged data sets (limit by type)	LISTCAT	LISTCAT
List names of cataloged data sets (limit by naming convention)	LISTDS	LISTDS
Catalog data sets	1. DEFINE, 2. ALLOCATE	DEFINE
List contents of data set	EDIT, LIST*	PRINT
Rename data set	RENAME	ALTER
Delete data set	DELETE	DELETE
Copy data set	COPY*	REPRO

TSO/E commands and subcommands

TSO/E commands which require a data set name (for example, Edit, DELete, XMIT) first search the current allocations to see if the data set is already allocated to the TSO/E session. If the data set name is already

allocated, it will be used by the command. If the data set name is not allocated, it will be allocated based on the standard catalog search order. Therefore, if a data set is desired that is not cataloged, you must use the ALLOCATE command to allocate it to the TSO/E session (see “ALLOCATE command” on page 8). This data set will then be used by all subsequent commands that use this data set name as one of the parameters. Conversely, if an uncataloged data set is allocated to the TSO/E session with the same name as a cataloged data set, and the cataloged data set is desired, you must first use the FREE command for the uncataloged data set so that the standard catalog search order will be used to find the cataloged data set.

Summary of TSO/E commands

Table 3: Summary of the TSO/E commands

Command	Function
ALLOCATE	Dynamically allocates data sets.
ALTLIB	Defines alternative application-level libraries of REXX EXECs or CLISTs.
ATTRIB	Builds a list of attributes for non-VSAM data sets.
CALL	Loads and executes a program.
CANCEL	Ends the processing of batch jobs submitted at your terminal.
DELETE	Deletes data set entries or members of a partitioned data set.
EDIT	Creates, modifies, stores, submits, retrieves, and deletes data sets. See command definitions for definitions of EDIT subcommands.
END	Ends a CLIST.
EXEC	Executes a CLIST or REXX exec.
EXECUTIL	Changes various characteristics that control how REXX execs run in the TSO/E address space only.
FREE	Releases previously allocated data sets, changes the output of a SYSOUT data set, deletes attribute lists, or changes data set disposition.
HELP	Gets information about the function, syntax, and operands of commands and subcommands and information about certain messages.
LINK	Invokes the linkage editor service program.
LISTALC	Lists data sets that are currently allocated to the TSO/E session.
LISTBC	Displays messages of general interest.
LISTCAT	Lists entries from a catalog by name or entry type.
LISTDS	Displays attributes of data sets.
LOADGO	Loads a compiled or assembled program into real storage and begins execution.
LOGOFF	Ends your terminal session.
LOGON	Starts your terminal session.
MVSSERV	Starts a session between an IBM Personal Computer and a host computer running TSO/E MVS™.
OUTDES	Creates or reuses dynamic output descriptors.

Table 3: Summary of the TSO/E commands (continued)

Command	Function
OUTPUT	Directs output from a job to your terminal or to a specific data set; deletes the output, changes output class, routes output to a remote workstation, or releases the output for a job for printing by the subsystem.
PRINTDS	Formats and prints data sets on any printer defined to JES.
PROFILE	Changes or lists your user profile.
PROTECT	Prevents unauthorized access to your non-VSAM data sets.
RECEIVE	Retrieves transmitted files and restore them to their original format.
RENAME	Changes the name of a non-VSAM cataloged data set, changes the member name of a partitioned data set, or creates an alias for a partitioned data set member.
RUN	Compiles, loads, and executes the source statements in a data set.
SEND	Sends a message to another terminal user or to the system operator.
SMCOPY	Copies all or part of a stream or data set to another stream or data set.
SMFIND	Locates a string of characters in a stream.
SMPUT	Places a string of characters in a stream.
STATUS	Displays the status of a job.
SUBMIT	Submits one or more batch jobs for processing.
TERMINAL	Lists or changes operating characteristics of your terminal.
TEST	Tests a program or command processor written in Assembler language.
TIME	Displays CPU and session time, total service units used, local time of day and date.
TRANSMIT	Sends information, such as a message or a copy of information in a data set, to another user in the network.
TSOEXEC	Invokes an authorized command from an unauthorized environment.
TSOLIB	Dynamically links to different versions of load module libraries from within a user's TSO/E session.
VLFNOTE	Notifies VLF that a change has been made to a partitioned data set or a non-partitioned data set.
WHEN	Tests return codes from programs invoked from an immediately preceding CALL or LOADGO command, and to take prescribed action if the return code meets a specified condition.

Note: Except for the DELETE and LISTCAT commands, TSO/E does not support generation data group (GDG) data sets.

ALLOCATE command

Use the ALLOCATE command or the ALLOCATE subcommand of EDIT (the subcommand's function and syntax are identical to the ALLOCATE command) to allocate dynamically the VSAM and non-VSAM data sets, and UNIX files required by a program that you intend to execute. Each UNIX file system data set contains zero or more UNIX files.

Each UNIX file system data set has a 44-byte data set name and must be capitalized and cataloged. A UNIX file has a mixed-case name of up to 250 characters.

There is no documented API for an UNIX file system data set. Users can access UNIX files with BSAM, QSAM, VSAM and UNIX calls.

You can specify data set attributes for non-VSAM data sets that you intend to allocate dynamically in several ways:

- Use the LIKE operand to obtain the attributes from an existing model data set (a data set that must be cataloged) whose data set attributes you want to use. You can override model data set attributes by explicitly specifying the desired attributes on the ALLOCATE command.
- Identify a data set and describe its attributes explicitly on the ALLOCATE command.
- Use the ATTRIB command to build a list of attributes. During the remainder of your terminal session, you can have the system refer to this list for data set attributes by specifying the USING operand when you enter the ALLOCATE command. The ALLOCATE command converts the attributes into the data control block (DCB) operands for data sets being allocated. If you code DCB attributes in an attribute-list and you refer to the attribute-list using the USING operand on the ALLOCATE command, any DCB attribute you code on the ALLOCATE command is ignored.
- With the Storage Management Subsystem (SMS) installed and active, use the DATACLAS operand. Your storage administrator might provide default data set attributes through the automatic class selection (ACS) routine. Using DATACLAS to define the data class for the data set makes specifying all the attributes unnecessary.

In this book, "with SMS" indicates that SMS is installed and is active. "Without SMS" indicates that SMS is not installed. Requesting space, in terms of a quantity of logical records, is device-independent and is particularly useful in conjunction with a system-determined BLKSIZE. This space can be obtained by omitting the BLKSIZE operand and coding LRECL, RECFM, and DSORG, or acquiring these from SMS DATACLAS.

Data sets with SMS

If your installation has the Storage Management Subsystem (SMS), and it is active, SMS allows you to more easily define new data sets by managing storage requirements for you. The storage administrator at your installation determines the data sets that are to be managed by SMS. The administrator writes the automatic class selection (ACS) routine that SMS uses to assign definitions or classes to a new data set.

SMS can manage the following types of data sets:

- Physical sequential data sets
- Partitioned data sets
- VSAM data sets
- Generation data group (GDG) data sets
- Temporary data sets
- Virtual input output (VIO) data sets

SMS does *not* manage the following types of data sets:

- Tape data sets
- ISAM data sets
- Sysout data sets
- Subsystem data sets
- TSO/E data sets coming from or going to a terminal
- In-stream data sets

SMS classes

With SMS, a new data set can have one or more of the following three classes:

- *Data class* contains the data set attributes related to the allocation of the data set, such as LRECL, RECFM, SPACE, and TRACKS.

- *Storage class* contains performance and availability attributes related to the storage occupied by the data set. A data set that has a storage class assigned to it is defined as an "SMS-managed" data set.
- *Management class* contains the data set attributes related to the migration and backup of the data set, such as performed by DFSMShsm, and the expiration date of the data set. A management class can be assigned only to a data set that also has a storage class assigned.

All of the preceding classes are defined by the storage administrator at your installation. The administrator writes the automatic class selection (ACS) routines that SMS uses to assign the classes to a new data set.

The DATACLAS, MGMTCLAS, and STORCLAS operands of the ALLOCATE command simplify the process of allocating a new data set. For example, assigning the DATACLAS operand to a data set keeps you from having to specify all the attributes of the data set on the ALLOCATE command. If you assign a storage class (STORCLAS) to a data set, you do not have to specify a volume serial number (VOLUME) or a unit type (UNIT).

If you do not specify DATACLAS, MGMTCLAS, and STORCLAS or the overriding attributes (DSORG, RECFM, LRECL, and so forth), the system assumes the defaults that the storage administrator defined through the ACS routines. The ACS routines can either change or retain the specified data set attributes. You can specify both a class attribute and an overriding attribute, such as DATACLAS and SPACE. The system uses SPACE as the storage value and the allocation attributes associated with the name specified on DATACLAS.

Note: You must explicitly allocate a new SMS-managed data set with a disposition of NEW.

Allocating non-SMS-managed data sets

With SMS, you can specify DATACLAS to allocate non-SMS-managed data sets. You cannot, however, use the STORCLAS and MGMTCLAS operands. STORCLAS and MGMTCLAS determine whether a data set is managed by SMS.

Allocating z/OS UNIX data sets

For z/OS UNIX, you can specify the following operands: PATH, PATHDISP, PATHMODE, PATHOPTS, DSNTYPE(HFS), and DSNTYPE(PIPE). For more information, see *z/OS TSO/E User's Guide*.

Determining the SYSOUT class

If you code both a SYSOUT and OUTDES operand, the class specified in the output descriptor is ignored if

- you explicitly specify a class on the SYSOUT operand, or
- you have a default sysout class defined in your RACF® TSO segment, in UADS, or via a logon exit.

For example, consider the following:

ALLOC FI(XX) SYSOUT OUTDES(OUT1) REU

- Code SYSOUT with a class suboperand (e.g. SYSOUT(A)) to allocate to the specified class.
- If a class is not specified on the SYSOUT operand, then the SYSOUT is allocated to the default class, determined as follows:
  - To the class defined as the default sysout class at logon. The default sysout class is set at logon time from:
    1. a logon exit, or
    2. a UADS SYSOUT class, for a UADS defined user, or
    3. from the RACF TSO segment SYSOUTCLASS, for a RACF defined user.
- If no logon sysout default was defined, allocation defaults it:
  - from an OUTPUT Descriptor, if specified, or
  - from a default message class, if no output descriptor.

Example:

A user is defined to RACF with a SYSOUT default class of B in his TSO segment. Assume there is no logon exit that sets any other logon SYSOUT value. So this user would have default sysout class B. Assume the user performs this allocation:

```
OUTDES OUT1 CLASS=E FORMS=XYZ1 REUSE
ALLOC FI(SYSUT2) SYSOUT OUTDES(OUT1) SHR
```

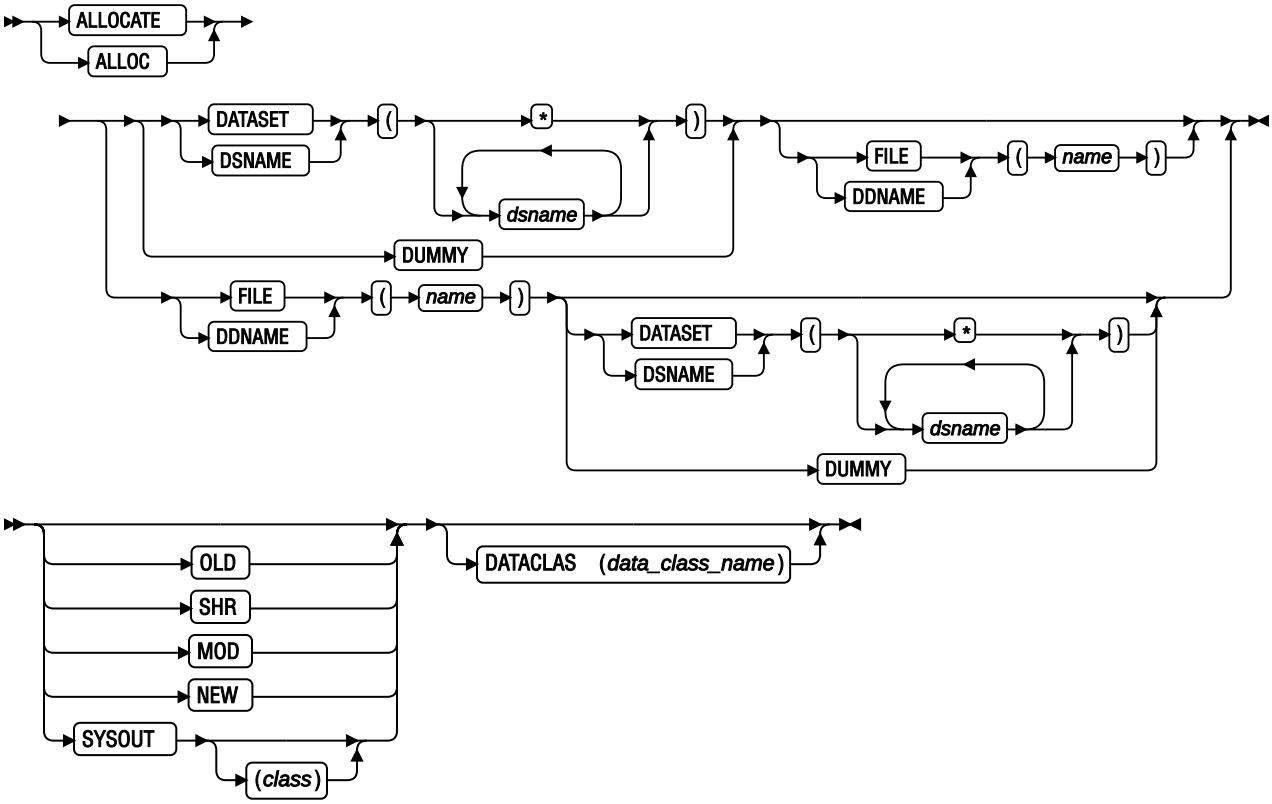
Because the user did not explicitly specify a SYSOUT class, SYSUT2 is allocated to sysout class B (the logon default from RACF). The class specified on the output descriptor OUT1 would be used only if no default sysout class exists.

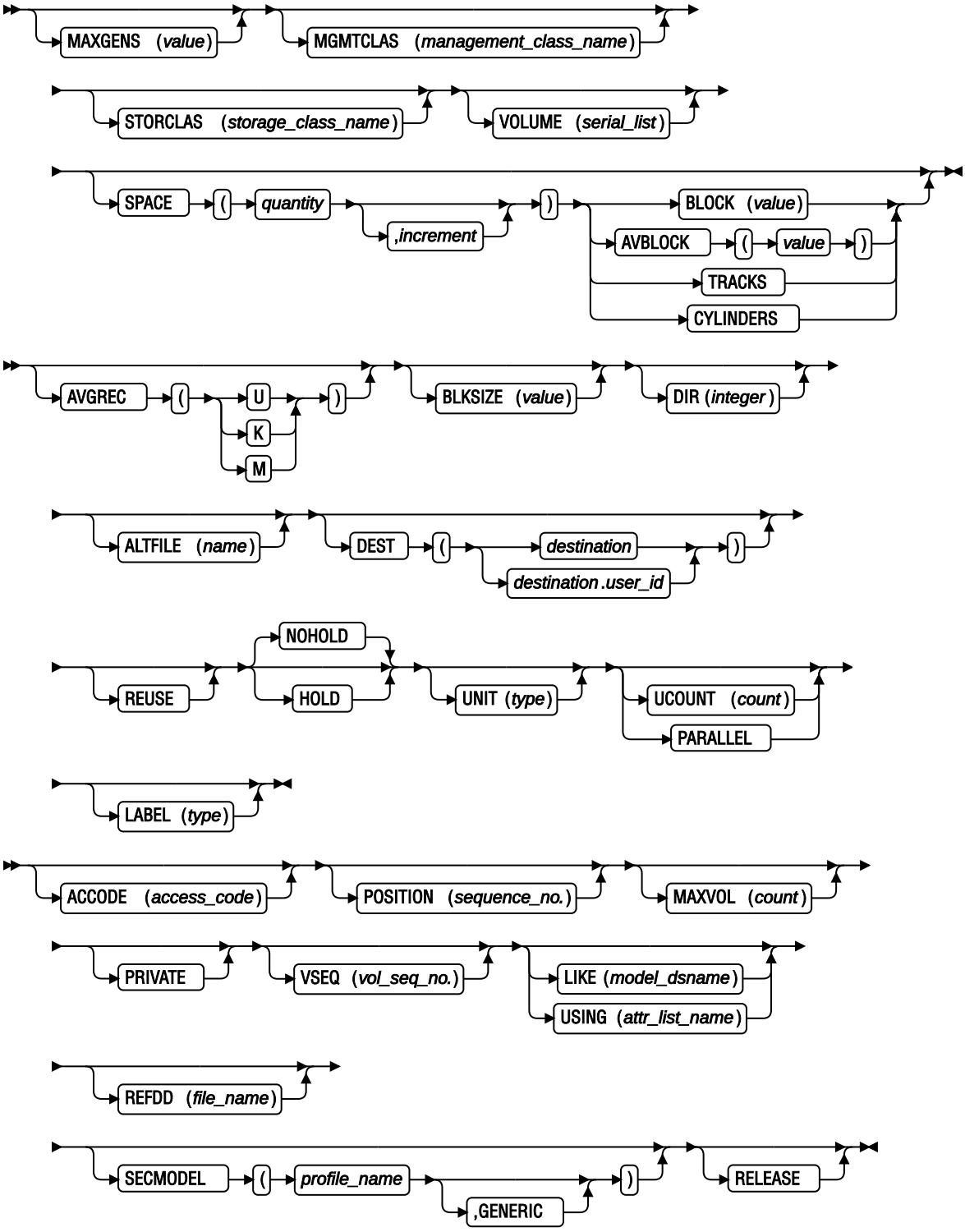
If the user needs to use class E, either explicitly specify SYSOUT(E) on the ALLOC command, or first have the RACF administrator delete the default class from the user's RACF TSO segment using:

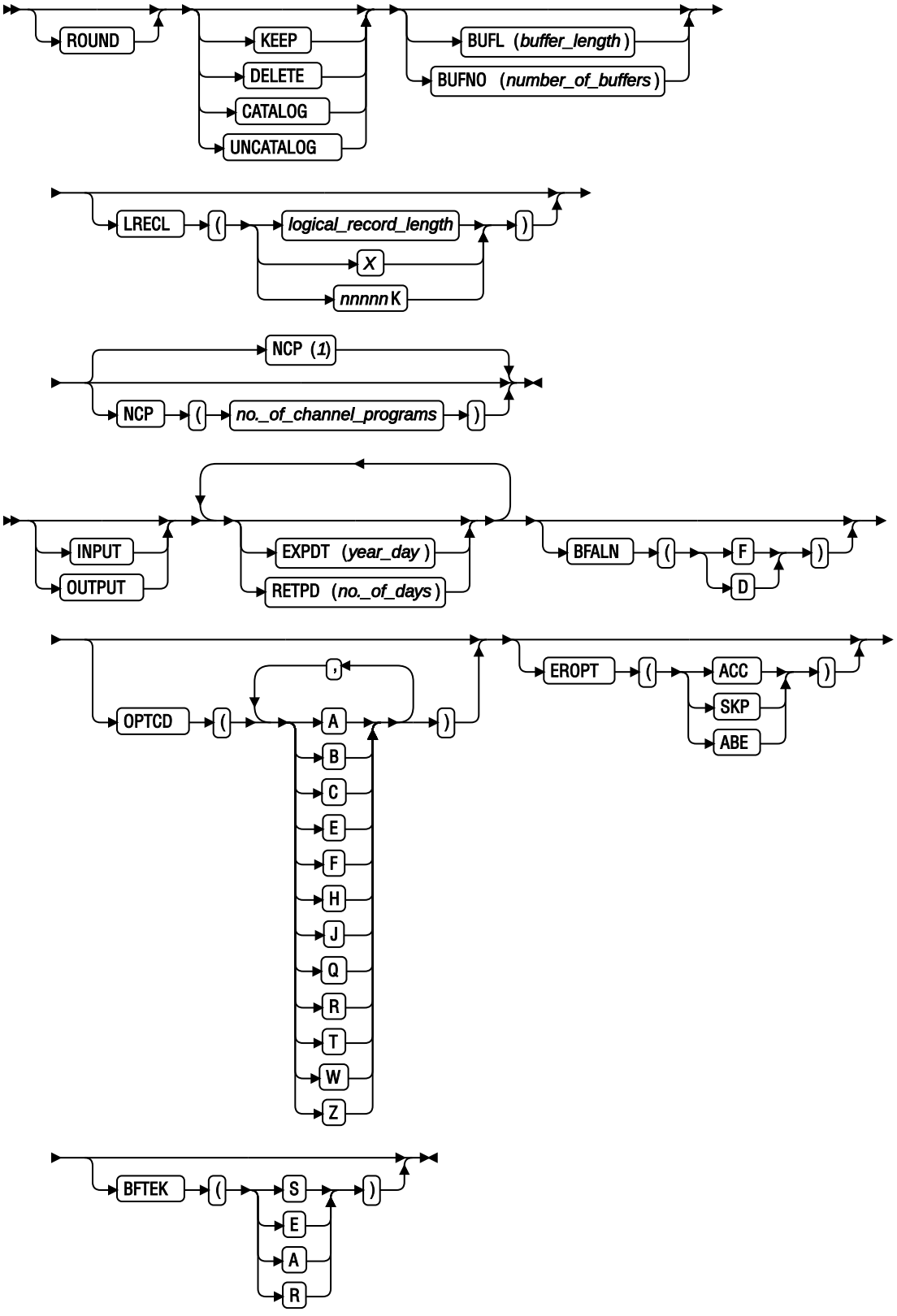
```
ALU userid TSO ( NOSYSOUT )
```

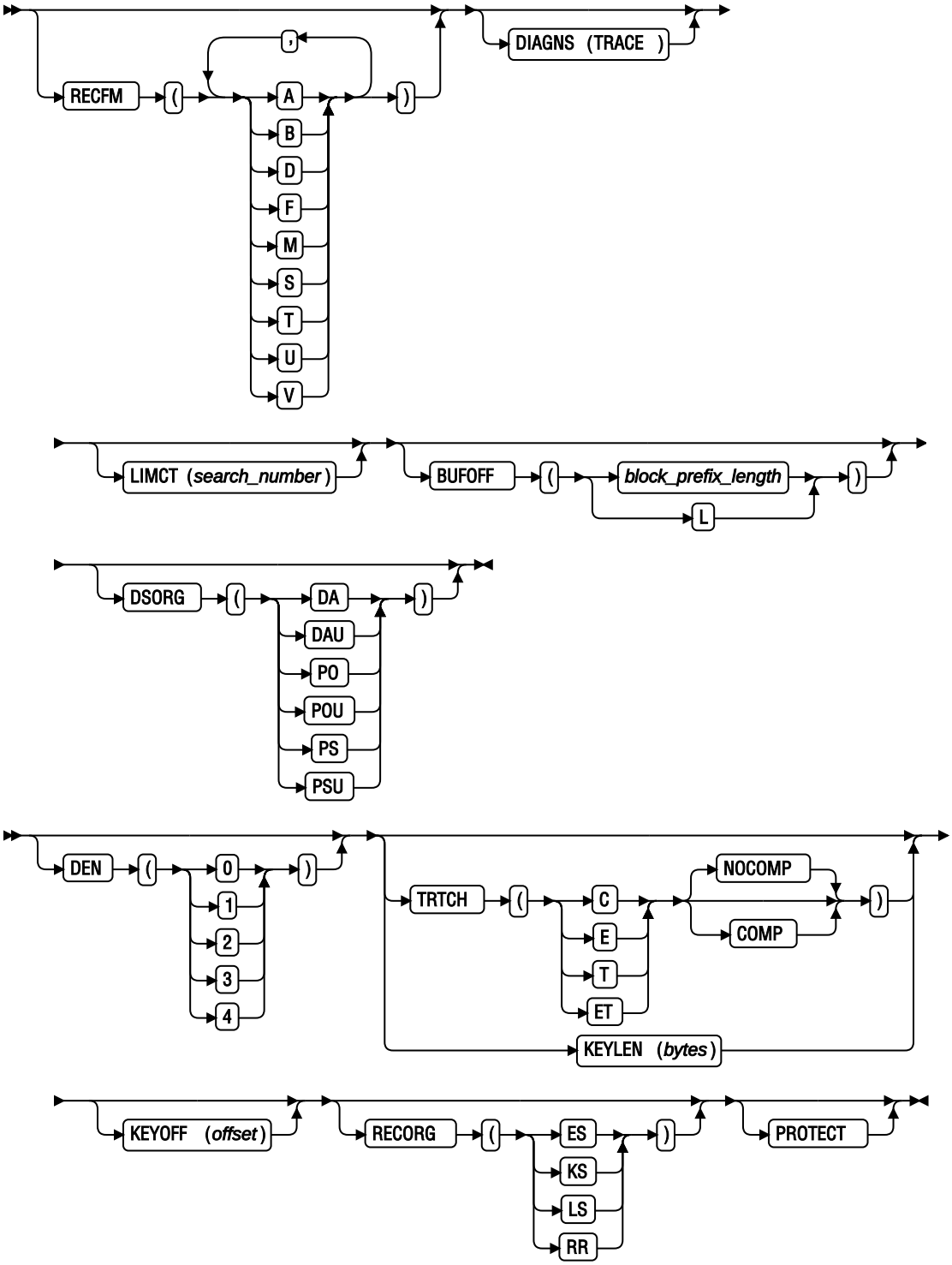
Then have the user LOGOFF and LOGON again to pick up the refreshed logon default with no sysout class. Now issuing the above OUTDES and ALLOC commands would allow the sysout default to be picked up from the output descriptor as sysout class E, since no logon default sysout class exists for this user.

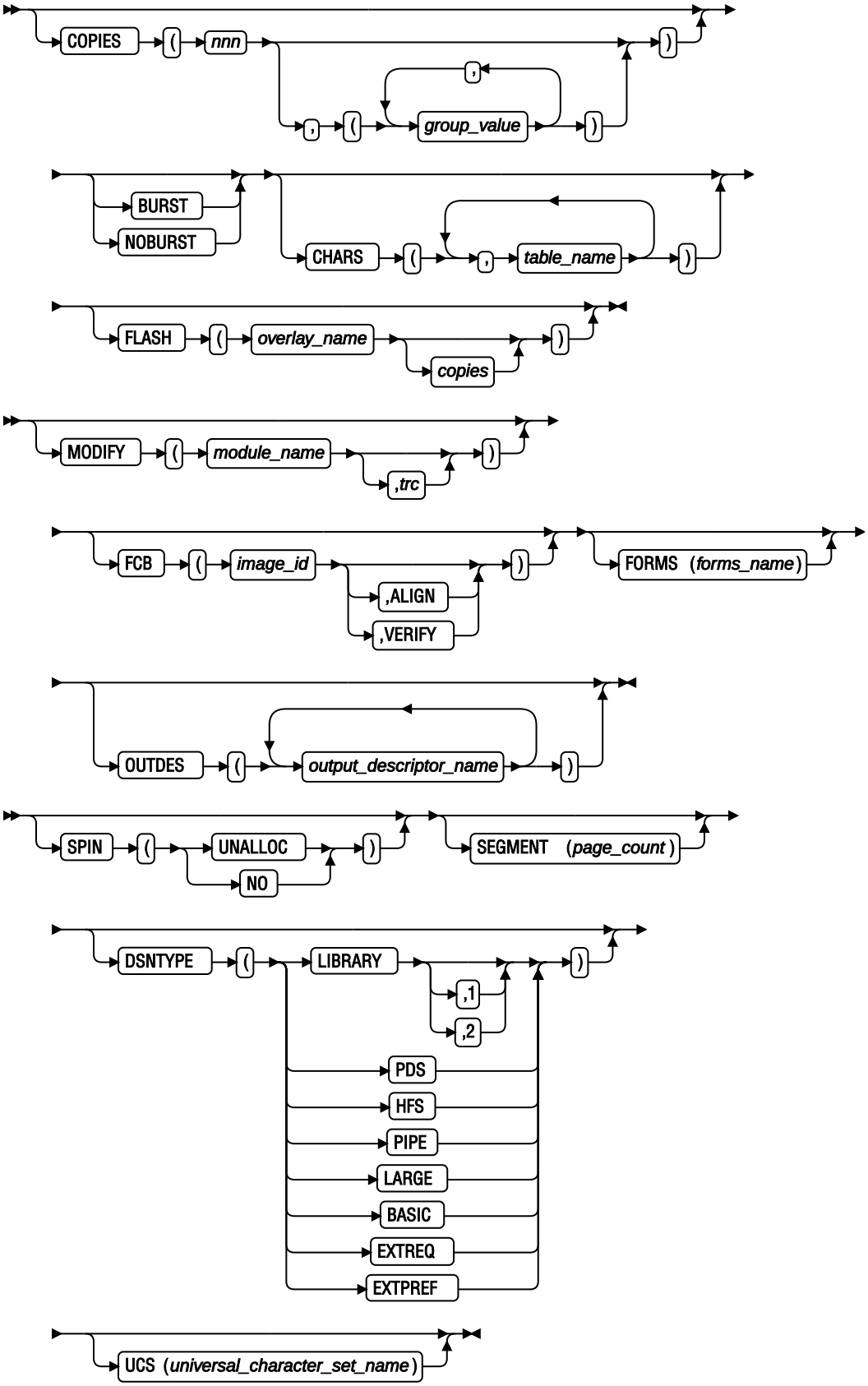
ALLOCATE command syntax

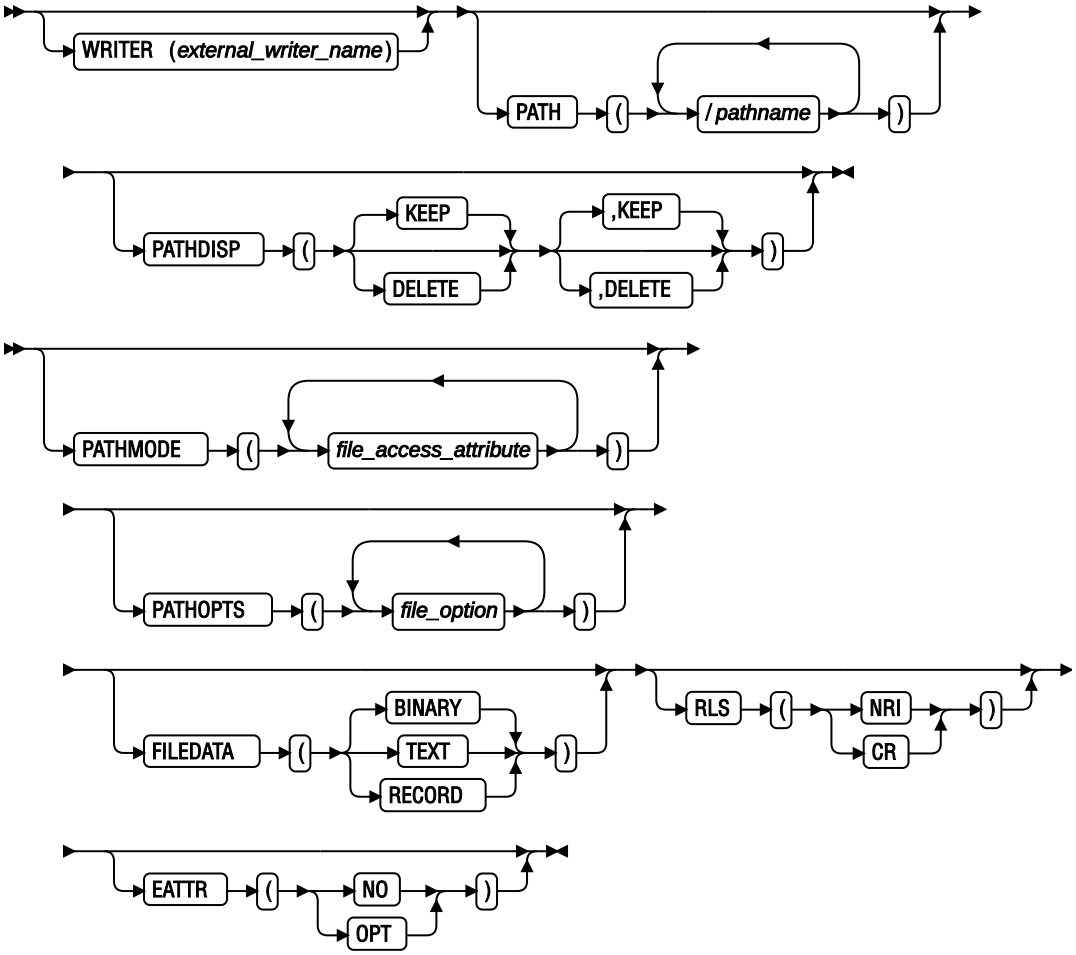



MAXGENS (value)
MGMTCLAS (management_class_name)
STORCLAS (storage_class_name)
VOLUME (serial_list)
SPACE
(
quantity
,increment
)
BLOCK (value)
AVBLOCK
(
value
)
TRACKS
CYLINDERS
AVGREC
(
U
K
M
)
BLKSIZE (value)
DIR (integer)
ALTFILE (name)
DEST
(
destination
destination.user_id
)
NOHOLD
REUSE
HOLD
UNIT (type)
UCOUNT (count)
PARALLEL
LABEL (type)
ACCODE (access_code)
POSITION (sequence_no.)
MAXVOL (count)
PRIVATE
VSEQ (vol_seq_no.)
LIKE (model_dsname)
USING (attr_list_name)
REFDD (file_name)
SECMODEL
(
profile_name
,GENERIC
)
RELEASE


ROUND
KEEP
DELETE
CATALOG
UNCATALOG
BUFL (buffer_length)
BUFNO (number_of_buffers)
LRECL
(
logical_record_length
X
nnnnnK
)
NCP (1)
NCP
(
no._of_channel_programs
)
INPUT
OUTPUT
EXPDT (year_day)
RETPD (no._of_days)
BFALN
(
F
D
)
OPTCD
(
,
A
B
C
E
F
H
J
Q
R
T
W
Z
)
EROPT
(
ACC
SKP
ABE
)
BFTEK
(
S
E
A
R
)


RECFM
(
,
A
B
D
F
M
S
T
U
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
T
ET
NOCOMP
COMP
)
KEYLEN (bytes)
KEYOFF (offset)
RECORG
(
ES
KS
LS
RR
)
PROTECT


COPIES
(
nnn
)
,
(
group_value
)
BURST
NOBURST
CHARS
table_name
FLASH
overlay_name
copies
MODIFY
module_name
,trc
FCB
image_id
,ALIGN
,VERIFY
FORMS (forms_name)
OUTDES
output_descriptor_name
SPIN
UNALLOC
NO
SEGMENT (page_count)
DSNTYPE
LIBRARY
,1
,2
PDS
HFS
PIPE
LARGE
BASIC
EXTREQ
EXTPREF
UCS (universal_character_set_name)



ALLOCATE command operands

DATASET(*dsname* | *) | DSNAME(*dsname* | *)

Specifies the name or a list of names of the data sets that are to be allocated. If a list of data set names is entered, ALLOCATE allocates and concatenates non-VSAM data sets. The data set name must include the descriptive (rightmost) qualifier and can contain a member name in parentheses.

If you specify a password, you are not prompted for it when you open a non-VSAM data set.

If you want to allocate a file to the terminal for input or output, only the following operands are processed:

```
ALLOCATE DA(*) FILE, DDNAME, BLOCK, BLKSIZE, USING
```

If you allocate more than one data set to your terminal, the block size and other data set characteristics, which default on the first usage, are also used for all other data sets. This happens for input or output. Use the ATTRIB command and the USING operand of ALLOCATE to control the data set characteristics.

- Data sets residing on the same physical tape volume cannot be allocated concurrently.
- The following items should be noted when using the concatenate function:
  - The data sets specified in the list must be cataloged. You can use the CATALOG operand of either the ALLOCATE or FREE commands to catalog a data set.
  - The maximum number of sequential data sets or partitioned members that you can concatenate is 255. The maximum in a partitioned concatenation is 255 PDS extents, PDSEs, or z/OS UNIX directories. For more information about the maximum number of partitioned data sets that you can concatenate, see *z/OS DFSMS Using Data Sets*. The data sets to be concatenated must all

have the same record format (RECFM). If you omit the BLKSIZE operand from the concatenation statement, the system uses the block size of the first data set. If the data sets have different block sizes, you must specify the data set with the largest block size first. In most situations, the access method automatically handles block size differences. For more information, see *z/OS DFSMS Using Data Sets*.

– The data set group is concatenated. You must free it to deconcatenate it. The file name that is specified for the FILE or DDNAME operand on the ALLOCATE command must be the same as that specified for the FILE or DDNAME operand on the FREE command.

– The system ignores all operands except for DATASET/DSNAME, FILE/DDNAME, and status operands. The following DCB attribute operands are allowed when concatenating data sets:

```
BLKSIZE   INPUT     EROPT     BUFOFF   USING
BUFL      OUTPUT    BFTEK     DEN
BUFNO     BFALN     DIAGNS    TRTCH
NCP       OPTCD     LIMCT     KEYLEN
```

- To allocate a member of a generation data group, specify its fully qualified data set name, including the absolute generation number in the low-level qualifier of the data set name. TSO/E ALLOCATE does not support DATASET names when specified with relative generation numbers. It only supports DATASET names when specified with the absolute generation numbers. (For example, if the most current generation number is 4, then the current GDG generation that is referenced by 'A.PARYOLL(0)' is to generation 4. However, since the relative generation format cannot be used, you might instead specify it with its absolute name, such as 'A.PAYROLL.G0004V00'.) For a definition of generation data set terminology, see the section about Generation Data Groups, in *z/OS DFSMS Using Magnetic Tapes*.
- The ALLOCATE command verifies the existence of a data set on the specified volume(s) only when the VOLUME operand is also specified.
- When you invoke ALLOCATE to perform dsname dynamic allocation, an "allocation environment" already exists for your request. It consists of the allocation requests, made through your JCL or internal dynamic allocation, that have not yet been deallocated. These resources are considered to be *existing allocations*, and are considered first in the attempt to fill your ALLOCATE requests.

  If possible, ALLOCATE uses an existing allocation to satisfy your dsname allocation request. Although some parameters can be changed if necessary, the request and the existing allocation must match according to several criteria before the allocation can be selected to satisfy your request.

  For more information about this criteria and using an existing allocation, see *z/OS MVS Programming: Authorized Assembler Services Guide*.

DUMMY
Specifies that no devices or external storage space are to be allocated to the data set, and no disposition processing is to be performed on the data set. Entering the DUMMY operand has the same effect as specifying NULLFILE as the data set name on the DATASET or DSNAME operand.

If you want to allocate a DUMMY data set, only the following operands are processed:

```
ALLOCATE DUMMY, FILE, DDNAME, BLOCK, BLKSIZE, USING
```

The following operands are not valid when you specify a DUMMY data set:

```
COPIES, DEST
```

FILE(*name*) | DDNAME(*name*)
Specifies the name to be associated with the data set. It can contain up to eight characters. (This name corresponds to the name on the data definition (DD) statement in job control language and must match the ddname in the data control block (DCB) that is associated with the data set.) For PL/I, this name is the file name in a DECLARE statement and has the form DCL file name FILE; for example, 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.

Do not use special ddnames unless you want to use the facilities that those names represent to the system.

For more information about the special ddnames SYSMDUMP, SYSUDUMP, SYSCHK, SYSCKEOV, and SYSABEND see *z/OS MVS JCL Reference*.

For more information about the special ddnames JOBCAT, JOBLIB, STEPCAT, and STEPLIB see *z/OS MVS JCL Reference*.

OLD | SHR | MOD | NEW | SYSOUT(*class*)

OLD
Indicates the data set currently exists and 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 operand is required.

SHR
Indicates the data set currently exists, but you do not require exclusive use of the data set. Others can use it concurrently. ALLOCATE assumes that 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 operand is required.

MOD
indicates that you want to append data to the end of the sequential data set. Do not catalog the data set or specify VOLUME=SER when you use DISP=MOD to create a new data set. After creation, the system changes the disposition of the data set to NEW. If the data set does not exist, a new data set is created and the disposition is changed to NEW. MOD data sets are retained by the system when you free them. The DATASET or DSNAME operand is required.

NEW
(non-VSAM only, unless SMS is running) indicates the data set does not exist and it is to be created. For new partitioned data sets, you must specify the DIR operand unless its data class provides a default value for it. For more information, see *z/OS DFSMSdfp Storage Administration*. If you specify a data set name, a NEW data set is kept and cataloged. If you do not specify a data set name, it is deleted when you free it or log off.

SMS manages data sets that were allocated with a disposition of NEW while SMS was active.

SYSOUT[(*class*)]
indicates that the data set is to be a system output data set. An optional subfield can be defined giving the output class of the data set. Output data is initially directed to the job entry subsystem (JES) and can 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.

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.

You can specify the OUTDES operand of the ALLOCATE command or the PRINTDS command to supply the name or names of the output descriptors that were created by the OUTPUT JCL statements in the LOGON procedure. Specifying OUTDES eliminates the need to supply information that is related to the printer or the type of printing to be done. For more information about establishing OUTPUT JCL statements in the LOGON procedure, see *z/OS TSO/E Customization*.

If you do not specify an output class value, the ALLOCATE command uses the default output class, which was determined during logon for your user ID. If no default class was set for your user ID,

JES assigns an output class according to its assignment procedures, using any referenced or default output descriptors.

For information about how a sysout class is chosen, especially when an output descriptor is specified, see "Determining the SYSOUT class" on page 10.

If you want to allocate a SYSOUT data set, the following operands are used exclusively with SYSOUT:

```
ALLOCATE DDNAME, SYSOUT, DEST, HOLD, NOHOLD, COPIES, BURST/NOBURST,
CHARS, FLASH, MODIFY, FCB, FORMS, OUTDES, UCS, WRITER, SPIN, SEGMENT
```

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:

- If the LIKE operand or any space operands (SPACE, DIR, BLOCK, BLKSIZE, AVBLOCK, TRACKS, or CYLINDERS) are specified, then the status defaults to NEW.
- If the COPIES operand is specified, then the status defaults to SYSOUT.
- If the DATASET/DSNAME operand is entered without the LIKE operand or any space operands, then the status defaults to OLD.
- If the LIKE operand, the DATASET/DSNAME operand, and the space operands are all omitted, you are prompted to enter a status value.

VOLUME(*serial_list*)

Specifies the serial number(s) of an eligible direct access volume(s) on which a new data set is to reside or on which an old data set is located. If you specify VOLUME 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). You can specify up to 255 volume serial numbers.

With SMS, the VOLUME operand is not suggested. The system determines the UNIT and VOLUME from the storage class (STORCLAS operand) associated with the data set. If SMS does not manage the data set and you want to allocate a data set to a *specific* volume, explicitly specify VOLUME.

DATACLAS(*data_class_name*)

If SMS is active, specifies the name, 1 to 8 characters, of the data class for the data set. The data set does not have to be managed by SMS.

Using the DATACLAS operand to define the data class makes specifying all the attributes for a data set unnecessary. For example, the storage administrator might provide RECFM, LRECL, RECORG, KEYLEN, and KEYOFF as part of the data class definition. However, you can override the DATACLAS operand by explicitly specifying the appropriate operands on the ALLOCATE command. If you specify DATACLAS for an *existing* data set, SMS ignores it.

The data class defines the following data set allocation attributes:

- Data set organization (record organization or record format):
  - Record organization (RECORG)
  - Record format (RECFM)
- Record length (LRECL)
- Key length (KEYLEN)
- Key offset (KEYOFF)
- Space allocation
  - AVGREC
  - SPACE
- Expiration date (EXPDT) or retention period (RETPD)
- Volume number (VOLUME)

- For VSAM data sets, the following:
  - IMBED or REPLACE
  - CISIZE
  - FREESPACE
  - SHAREOPTIONS

Note: Without SMS, the system syntax checks and then ignores the DATACLAS operand.

MAXGENS(*value*)
Specifies the maximum number of generations for members in a Version 2 PDSE.

The value is 0 to 2,000,000,000. The default is 0.

The value may be limited by MAXGENS_LIMIT in the IGDSMSxx member of PARMLIB.

MGMTCLAS(*management_class_name*)
With an SMS-managed data set, specifies the name, 1 to 8 characters, of the management class for a new data set. When possible, do not specify MGMTCLAS. Instead, use the default your storage administrator provides through the ACS routines.

After the data set is allocated, attributes in the management class control the following:

- The migration of the data set, which includes migration from primary storage to Data Facility Storage Management Subsystem Hierarchical Storage Manager (DFSMShsm) owned storage to archival storage.
- The backup of the data set, which includes frequency of backup, number of versions, and retention criteria for backup versions.

Note: Without SMS, the system syntax checks and then ignores the MGMTCLAS operand.

STORCLAS(*storage_class_name*)
With SMS, specifies the name, 1 to 8 characters, of the storage class. If you have no specific storage class requirements, do not specify STORCLAS. Instead, use the default your storage administrator provides through the ACS routines.

The storage class replaces the storage attributes that are specified on the UNIT and VOLUME operand for non-SMS-managed data sets.

An "SMS-managed data set" is defined as a data set that has a storage class assigned. A storage class is assigned when the installation-written ACS routine selects a storage class for the new data set.

Note: Without SMS, the system syntax checks and then ignores the STORCLAS operand.

SPACE(*quantity*,*increment*)

Specifies the amount of space to be allocated when creating or extending a DASD data set.

quantity
Specifies the number of units of space to be allocated initially when creating or extending a DASD data set.

increment
Specifies the number of units of space to be added to the data set each time the previously allocated space has been filled. You must specify the primary quantity along with the increment value.

SPACE can be specified for SYSOUT, NEW, and MOD data sets. The SPACE parameter has no effect if SYSOUT is coded also.

If you omit this operand, the system uses the first of the following sources that provides values:

- Your installation might change the value for SPACE or set a default by using the IEFDB401 exit routine. See *z/OS MVS Installation Exits*.
- A data class can specify space values. You can specify a data class name with the DATACLAS keyword or SMS can provide one. The data set does not have to be SMS-managed. In addition, your storage administrator can use SMS to override the data class name that you code.

- Your installation might have set a default for SPACE using the ALLOCxx PARMLIB member. See *z/OS MVS Initialization and Tuning Reference*.
- If none of the above sources provide a value for SPACE, the IBM-supplied default is SPACE(4,24) AVBLOCK(8192).

With SMS, the system does not prompt you for the space. To have the system obtain the amount of space, specify both the AVGREC and AVBLOCK operand.

Specifying AVGREC requires you to also specify an average record length. You can use the AVBLOCK keyword. If you do not specify BLOCK or BLKSIZE, the system determines the optimized value.

When you specify SPACE, you must specify a unit of space. To indicate the unit of space for allocation, you must specify one of the following:

- BLOCK(value)
- 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. With SMS, if you do not specify BLKSIZE, the system determines an optimum DCB block size for the new data set.
- AVBLOCK(value): Multiply the value of the AVBLOCK operand by the quantity value of the SPACE operand. The AVBLOCK is the average logical record length and should be coded with the AVGREC(U, K, or M) 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.

See the preceding information concerning the AVGREC operand about how the amount of space is determined for each of these keywords.

BLOCK(*value*)

Specifies the average length of the blocks written to the data set. The maximum block value used to determine space to be allocated is 65,535. The block value is the unit of space used by the SPACE operand. A track or a cylinder on one device can represent a different amount of storage (number of bytes) than a track or a cylinder on another device. The unit of space value is determined in one of the following ways:

- From the default value, which is SPACE (4,24) AVBLOCK (8192), when no space operands (that is, SPACE, BLOCK, TRACKS, AVBLOCK, or CYLINDERS) are specified.
- From the BLOCK operand, if specified.
- From the model data set, if the LIKE operand is specified and BLOCK, TRACKS, AVBLOCK, or CYLINDERS are not specified on ALLOCATE. This is true only when SMS is inactive. When SMS is active, LIKE does not retrieve the unit of space(CYL/TRK/BLK) from the model data set.
- From the BLKSIZE operand, if BLOCK is not specified.

Note that the default value for space is installation dependent. Your installation might have changed the default value.

If you do not specify BLKSIZE, the system attempts to determine an optimum DCB block size for the new data set.

AVBLOCK(*value*)

Specifies the average length (in bytes) of the records that are written to the data set. This parameter only has an effect if SPACE is specified.

With SMS, to allocate space in a quantity of records instead of blocks, tracks, or cylinders, use both the AVBLOCK and AVGREC operands. Do not code the BLOCK, TRACKS, or CYLINDERS operands.

TRACKS
Specifies the unit of space is to be a track. This parameter only has an effect if SPACE is specified.

With SMS, if you do not want to explicitly specify TRACKS, specify both the AVGREC and AVBLOCK operands instead of the TRACKS operand.

CYLINDERS
Specifies the unit of space is to be a cylinder. This parameter only has an effect if SPACE is specified.

With SMS, if you do not want to explicitly specify CYLINDERS, specify both the AVGREC and AVBLOCK operands instead of the CYLINDERS operand.

AVGREC(U | K | M)

Together with AVBLOCK in SMS, determines the size of the average record length. This parameter only has an effect if SPACE is specified. Following are the values for AVGREC:

U
Use the primary and secondary space quantities specified on the SPACE operand.

K
Multiply primary space quantity and secondary space quantity specified on the SPACE operand by 1024 (1 K).

M
Multiply primary space quantity and secondary space quantity specified on the SPACE operand by 1,048,576 (1 M).

For example, if you want to allocate 12 mega units of space, you can specify SPACE(12) AVGREC(M), which results in 12 * 1,048,576 = 12,582,912.

To get a secondary space quantity, you need to specify SPACE(12,1) AVGREC(M). This specification provides 12 mega units of primary space and 1 mega unit of secondary space. The unit of space is determined by either BLOCK, BLKSIZE, or AVBLOCK.

If AVGREC(K), AVBLOCK(128), and SPACE(5,2) are specified, the average record length is 128, the primary quantity of records is 5K, and the second quantity of records is 2K.

BLKSIZE(*blocksize*)
Specifies the block size for the data set. The maximum allowable decimal value for block size recorded in the DCB is 32,760.

With DASD, labeled tape or spooled data set, or a TSO terminal, if you do not specify BLKSIZE, the system determines the optimum block size for the new data set unless you have undefined length records. For more information see *z/OS DFSMS Using Data Sets*.

The DCB block size is determined in one of the following ways:

- If USING is specified, from the attribute list. You cannot use the BLKSIZE operand on ALLOCATE for the block size.
- If you specify BLKSIZE on ALLOCATE, from the BLKSIZE operand.
- If LIKE is specified and BLKSIZE is not specified on ALLOCATE, from the model data set.

  With SMS, BLKSIZE is *not* copied from the model data set. Without SMS, BLKSIZE is copied from the model data set.
- If neither USING, BLKSIZE, nor LIKE is specified, from the BLOCK operand.

The block size that you specify to be recorded in the data control block (DCB) must be consistent with the requirements of the RECFM operand:

- RECFM(F) – the block size must be equal to the logical record length.
- RECFM(F,B) – the block size must be an integral multiple of the logical record length.

- RECFM(V) – 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) – the block size must be equal to or greater than the largest block in the data set. 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 go into the block. Each logical record length must allow space for a four-byte record descriptor word. Because the number of logical records can vary, you must estimate the optimum block size and the average number of records for each block based on your knowledge of the application that requires the I/O.
- RECFM(U) – for files allocated to the TSO/E terminal with RECFM(U) and BLKSIZE(80), one character is truncated from the line. That character (the last byte) is reserved for an attribute character.

  Specify BLKSIZE with the ALLOCATE command when using the LIKE operand, because optimal BLKSIZE is not determined by the system for a RECFM(U) data set.

The operands BLOCK, BLKSIZE, AVBLOCK, TRACKS, and CYLINDERS can be specified for SYSOUT, NEW, or MOD data sets. The operands 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. Generally it is not useful for a PDSE.

ALTFILE(*name*)
Specifies the name associated with the SYSIN subsystem data set that is to be allocated. It can contain up to 8 characters. This operand is used primarily in the background.

DEST({*destination* | *destination*.*user_id*})
Specifies a specific remote workstation or a user at a specific remote workstation to which SYSOUT data sets are directed upon deallocation. Specify 1 to 8 characters for either the destination or the user ID.

REUSE
Specifies the file name being allocated is to be freed and reallocated if it is currently in use.

When you allocate a data set with file name or ddname, give it a disposition of SHR or OLD. You cannot use the REUSE operand to reallocate a file from a disposition of OLD to a disposition of SHR. However, you can first free the file with a disposition of OLD, then reallocate it with a disposition of SHR.

HOLD | NOHOLD

HOLD
Specifies the data set is to be placed on a HOLD queue upon deallocation.

NOHOLD
Specifies processing of the output should be determined by the HOLD/NOHOLD specification associated with the particular SYSOUT class specified. However, the specification associated with the SYSOUT class can be overridden by using the NOHOLD operand on the FREE command.

UNIT(*type*)
Specifies the type of the unit to which a file or data set is to be allocated. You can specify an installation-defined group name, a generic device type, or a specific device number.

This distinguishes numeric-only device numbers from generic device types that contain only four-character numerics.

If volume information is not supplied (volume and unit information is retrieved from a catalog), the unit type that is coded overrides the unit type from the catalog.

If the data set is managed by SMS, the UNIT operand is not suggested. The system determines the UNIT and VOLUME from the storage class associated with the data set. If the storage administrator

has set up a default unit type under SMS regardless of whether the data set is SMS-managed, you do not have to specify UNIT.

Without SMS, if you do not specify UNIT, the default UNIT is obtained from the user attribute data set (SYS1.UADS) or the security system being used (if SYS1.UADS is not being used).

The default specification for the UNIT operand relates to the LOGON procedure selected in the foreground. If the ALLOCATE command is to be executed in the background, and the UNIT operand is not specified, the default operand value is *not* obtained from the user attribute data set (SYS1.UADS) or the security system. See the *z/OS TSO/E User's Guide*, for a description of command processing differences when executing foreground commands from a background job.

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

PARALLEL
Specifies one device is to be mounted for each volume specified on the VOLUME operand or in the catalog. This is meaningful only for magnetic tape.

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

ACCODE(*access_code*)
Specifies or changes the accessibility code for an ISO/ANSI labeled output tape data set. The purpose of the code is to protect the ANSI data set from unauthorized use. Up to 8 characters (A-Z) are permitted in the access code, but only the first character is validated by ANSI. The first character must be an uppercase alphabetic character. An installation exit routine validates it. That routine is described in *z/OS MVS Installation Exits*.

POSITION(*sequence_no.*)
Specifies the relative position (1- 65535) of the data set on a multiple data set tape. The sequence number corresponds to the data set sequence number field of the label operand in JCL.

MAXVOL(*count*)
Specifies the maximum number of volumes that the data set can reside upon. For DASD, the maximum value is 59. For magnetic tapes, the maximum value is 255. This number corresponds to the count field on the VOLUME operand in JCL.

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

If VOLUME and PRIVATE operands are not specified and the value specified for MAXVOL exceeds the value specified for UCOUNT, the system does 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 demounts one of the volumes and mounts 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 operand in JCL.

LIKE(*model_dsname*)
Specifies the name of an existing model data set whose attributes are to be used as the attributes of the new data set being allocated. This data set must be cataloged and must reside on a direct access device. The volume must be mounted when you issue the ALLOCATE command.

If SMS is active in the system, ALLOCATE assigns attributes to a new data set by copying all of the following attributes from the model data set:

- Primary space quantity (SPACE)
- Secondary space quantity (SPACE)
- Space unit (BLOCK, AVBLOCK, TRACKS, CYLINDERS)
- AVGREC unit (KB, megabyte)

- Directory space quantity (DIR)
- Data set organization:
  - RECORG for a VSAM data set
  - DSORG for a non-VSAM data set
- Logical record length (LRECL)
- Key length (KEYLEN)
- Record format (RECFM)
- Key offset (KEYOFF)
- Data set type (DSNTYPE)
- Extended attribute status (EATTR)

Note, however, that if SMS is active, the following attributes are *not* copied:

- Optional services code (OPTCD) - for ISAM data sets only
- Block size (BLKSIZE)
- Volume sequence number (VSEQ)
- Data set expiration date (EXPDT)

You can use the LIKE operand even if none of your existing data sets have the exact attribute values you want to use for a new data set. You can override attributes copied from a model data set by specifying the LIKE operand and the operands corresponding to the attributes you want to override on the ALLOCATE command.

The following items should be considered when using the LIKE operand:

- NEW is the only valid data set status that can be specified with the LIKE operand.
- The LIKE operand must be specified with the DATASET operand.
- Only one data set name can be specified on the DATASET/DSNAME operand.
- With SMS, block size is *not* copied from the model data set. If you do not specify the block size, the system determines the optimal block size for the data set, unless the data set has RECFM(U).

  The attributes copied from the model data set override attributes from the data class.
- If the new data set to be allocated is specified with a member name, indicating a partitioned data set (PDS), then you are prompted for directory blocks unless that quantity is explicitly specified on the ALLOCATE command or defaulted from the LIKE data set.

  If the new data set name is specified with a member name, but the model data set is sequential and you have not explicitly specified the quantity for directory blocks, then you are prompted for directory blocks.
- If you specify the directory value as zero and the model data set is a partitioned data set, then the new data set is allocated as a sequential data set.
- Unless you explicitly code the SPACE operand for the new data set, the system determines the space to be allocated for the new data set by adding up the space allocated in the first three extents of the model data set. Therefore, the space allocated for the new data set will generally not match the space that was specified for the model data set. Also, the system allocates the space for the new data set in tracks.
- Without SMS, the DSNTYPE keyword must be specified to allocate a PDSE data set.

USING(*attr_list_name*)

Specifies the name of a list of attributes that you want to have assigned to the data set you are allocating. The attributes in the list correspond to, and are used for, data control block (DCB) operands. (Note to users familiar with batch processing: These DCB operands are the same as those normally specified by using 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.

USING, LIKE, and REFDD are mutually exclusive.

Note: You cannot specify the DCB operands (operands that are also on the ATTRIB command) with the USING operand.

REFDD(*file_name*)

If SMS is active, specifies the ddname of an existing data set whose attributes are copied to the new data set. The following attributes are copied to the new data set:

- Data set organization (record organization or record format):
  - Record organization (RECORG)
  - Record format (RECFM)
- Directory space quantity (DIR)
- Record length (LRECL)
- Key length (KEYLEN)
- Key offset (KEYOFF)
- Space allocation:
  - AVGREC
  - SPACE
  - TRACK, CYLINDER, BLOCK

When you allocate a data set with REFDD, specify a disposition of NEW. For example,

```
alloc da('user1.my.text') fi(dd1) shr reu
alloc f(dd2) da('user2.your.data') new refdd(dd1)
```

USER1.MY.TEXT is an existing and cataloged data set. Note that the block size (BLKSIZE) is *not* copied to the new data set USER2.YOUR.DATA.

The retention period (RETPD) or expiration date (EXPDT) is not copied to the new data set.

The LIKE, REFDD, and USING operands are mutually exclusive.

Note: Without SMS, the system syntax checks and ignores the REFDD operand.

SECMODEL(*profile_name*[,GENERIC])

Specifies the name of an existing RACF data set profile, the attributes of which are copied to the discrete profile. Use SECMODEL when you want a different RACF data set profile than the default profile selected by RACF, or when there is no default profile. The model profile can be one of the following profiles:

- RACF model profile
- RACF discrete data set profile
- RACF generic data set profile

GENERIC identifies that the profile name is a generic data set profile. For example, if you want to create a generic data set profile, specify SECMODEL(*profile_name*,GENERIC).

The following information from the RACF data set profile is copied to the discrete data set profile of the new data set:

- OWNER indicates the user or group assigned as the owner of the data set profile.
- ID indicates the access list of users or groups authorized to access the data set.
- UACC indicates the universal access authority associated with the data set.
- AUDIT/GLOBALAUDIT indicates which access attempts are logged.

- ERASE indicates that the data set is to be erased when it is deleted (scratched).
- LEVEL indicates the installation-defined level indicator.
- DATA indicates installation-defined information.
- WARNING indicates that an unauthorized access causes RACF to issue a warning message, but allows access to the data set.
- SECLEVEL indicates the name of an installation-defined security level.

Note: Without SMS, the system syntax checks and ignores the SECMODEL operand.

For more information about RACF, see *z/OS Security Server RACF Command Language Reference*.

RELEASE
Specifies unused space is to be deleted when the data set is closed.

If you use RELEASE for a new data set with the BLOCK or BLKSIZE operand, then you must also use the SPACE operand.

ROUND
Specifies 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 operand on the SPACE parameter in JCL.

KEEP | DELETE | CATALOG | UNCATALOG

KEEP [1]
Specifies the data set is to be retained by the system after it is freed. If the data set is SMS-managed, KEEP has the same effect as CATALOG.

DELETE [1]
Specifies the data set is to be deleted after it is freed. If the data set is SMS-managed, DELETE also forces UNCATALOG.

CATALOG [1]
Specifies the data set is to be retained by the system in a catalog after it is freed.

UNCATALOG [1]
Specifies the data set is to be removed from the catalog after it is freed. If the data set is not SMS-managed and you do not want the system to retain the data set, you must also specify the DELETE operand.

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 are 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 can be limited to a smaller number of buffers depending on the amount of available virtual storage. 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 if the program was designed to use buffers obtained during OPEN.

[1] A command processor can modify the final disposition of this operand.

(4) Automatically with QSAM	(4) You may omit BUFNO and accept the system default, which is five or one, except with an extended format data set. For more information see *z/OS DFSMS Using Data Sets*.

LRECL({*logical_record_length* | X | *nnnnn*K})
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.

If SMS is active, you can use the DATACLAS operand in place of LRECL to specify the logical record length. If you specify LRECL, the system determines the block size.

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 field 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.

LRECL(*nnnnn*K) allows users of ISO/ANSI extended logical records and QSAM locate mode users to specify a K multiplier on the LRECL operand. *nnnnn* can be a number within 1-16,384. The K indicates that the value is multiplied by one thousand and twenty-four (1024).

For variable-length spanned records (VS or VBS) processed by QSAM (locate mode) or BSAM, specify LRECL (X) when the logical record exceeds 32756 bytes.

NCP(*number_of_channel_programs*)
Specifies the maximum number of READ or WRITE macro instructions allowed before a CHECK or WAIT macro instruction is issued. The maximum number must not exceed 255 and must be less than 255 if the address space does not have enough virtual storage. 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 a BSAM data set opened for INOUT or a BDAM data set opened for UPDAT is to be processed for input only. This operand overrides the INOUT (BSAM) option or UPDAT (BDAM) option in the OPEN macro instruction to INPUT. This is useful if you only have READ access authority to the data set.

OUTPUT
Specifies a BSAM data set opened for OUTIN or OUTINX is to be processed for output only. This operand overrides the OUTIN option in the OPEN macro instruction to OUTPUT or the OUTINX option in the OPEN macro instruction to EXTEND.

EXPDT(*year_day*)
Specifies the data set expiration date. Specify the year and day in one of two forms:

1. *yyddd*, where *yy* is the last two-digit number for the year and *ddd* is the three-digit number for the day of the year. The maximum value for the year is 99 (for 2099). The minimum value for the day is 000 and the maximum value is 366.
2. *yyyy/ddd*, where *yyyy* is the four-digit number for the year and *ddd* is the three-digit number for the day of the year. The slash is required. The maximum value for the year is 2155. The minimum value for the day is 000 and the maximum value is 366.

EXPDT is mutually exclusive with RETPD.

If SMS is active, the expiration date might have been defined by the DATACLAS operand.

RETPD(*number_of_days*)
Specifies the data set retention period in days. The value can be a five-digit decimal number with a current maximum value of 93000.

RETPD is mutually exclusive with EXPDT.

BFALN({F | D})
Specifies the boundary alignment of each buffer as follows:

F
Each buffer starts on a fullword boundary that might not be a doubleword boundary.

D
Each buffer starts on a doubleword boundary.

If you do not specify this operand, the system defaults to a doubleword boundary.

OPTCD(A, B, C, E, F, H, J, Q, R, T, W, and Z or all)
Specifies the following optional services that you want the system to perform. For a detailed discussion of these services, see the OPTCD subparameter of the DCB parameter in *z/OS MVS JCL Reference* and *z/OS DFSMS Macro Instructions for Data Sets*.

A
Specifies the actual device addresses be presented in READ and WRITE macro instructions.

B
Specifies the 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 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 embedded VSE checkpoint records on tape.

J
Specifies the character after the carriage control character is the table reference character for that line. The table reference character tells TSO/E which character arrangement table to select when printing the line.

Q
Requests the system to translate a magnetic tape from ASCII to EBCDIC or from EBCDIC to ASCII.

R
Requests the use of relative block addressing.

T
Requests the use of the user totaling facility.

W
Requests the system to perform a validity check when data is written on a direct access device.

Z
Requests the control program to shorten its normal error recovery procedure for input on magnetic tape.

You can request any or all of the services by combining the values for this operand. You can 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 to execute if an error occurs when a record is read or written. The options are:

ACC
To accept the block of records in which the error was found.

SKP
To skip the block of records in which the error was found.

ABE
To end the task abnormally.

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

S
Simple buffering.

E
Exchange buffering.

A
Automatic record area buffering.

R
Record buffering.

RECFM(A, B, D, F, M, S, T, U, and/or V)
Specifies the format and characteristics of the records in the data set. The format and characteristics must be completely described by one source only. If they are not available from any source, the default is an undefined-length record. For a discussion of the formats and characteristics of the RECFM subparameter of the DCB parameter, see *z/OS MVS JCL Reference*.

Use the following values with the RECFM operand:

A
Indicates the record contains ASCII printer control characters.

B
Indicates the records are blocked.

D
Indicates variable-length ASCII records.

F
Indicates the records are of fixed-length.

M
Indicates the records contain machine code control characters.

S
Indicates, 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 might span more than one block. Exchange buffering, BFTEK(E), must not be used.

T
Indicates the records can be written onto overflow tracks, if required. Exchange buffering, BFTEK(E), or chained scheduling, OPTCD(C), cannot be used.

U
Indicates the records are of undefined-length.

V
Indicates the records are of variable-length.

You can specify one or more values for this operand; at least one is required. If you use more than one value, you must separate each value with a comma or a space.

With SMS, the record format for a new data set might have been defined by the DATACLAS operand.

RECFM is mutually exclusive with RECORG.

DIAGNS(TRACE)
Specifies the Open/Close/EOV trace option that gives a module-by-module trace of the Open/Close/EOV work area and your 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 specifies 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

When you allocate a new data set and you do not specify the DSORG operand, DSORG defaults to partitioned organization (PO) if you specify a nonzero value for the DIR operand. If you do not specify a value in the DIR operand, the system assumes that you want a physical sequential (PS) data set. Note that the system does not store this default DSORG information into the data set until a program opens and writes to the data set. For more information about data set organization, see *z/OS MVS Programming: Authorized Assembler Services Guide*.

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}, {COMP | NOCOMP})
Specifies the recording technique for 7 or 18 track tape as follows:

C
Data **c**onversion with odd parity (the default) and no translation (the default).

E
Even parity with no translation (the default) and no conversion (the default).

T
Odd parity (the default) and no conversion (the default). BCD to EBCDIC **t**ranslation when reading and EBCDIC to BCD translation when writing.

ET
Even parity, and no conversion (the default). BCD to EBCDIC **t**ranslation when reading and EBCDIC to BCD translation when writing.

COMP| NOCOMP
Specifies whether data sets are to be compressed with IDRC to save space in tape.

This operand is mutually exclusive with KEYLEN.

KEYLEN(*bytes*)
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 for a record organization of physical sequential (PS) or partitioned (PO).

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 operand is mutually exclusive with TRTCH.

If SMS is active, the key length might have been defined by the DATACLAS operand. If you want to override it, explicitly specify KEYLEN. The number of bytes is as follows:

- 1 to 255 for a record organization of key-sequenced (RECORG(KS)).
- 0 to 255 for a record organization of physical sequential (PS) or partitioned (PO).

KEYOFF(*offset*)
If SMS is active, specifies the key position (offset) of the first byte of the key in each record. If you want to specify key offset or override the key offset defined in the data class (DATACLAS) of the data set, use KEYOFF. Specify KEYOFF only for a VSAM key-sequenced data set (RECORG(KS)).

Note: Without SMS, the system syntax checks and then ignores the KEYOFF operand.

RECORG({ES | KS | LS | RR})
If SMS is active, specifies the organization of the records in a new VSAM data set. If you want to override the record organization defined in the data class (DATACLAS) of the data set, use RECORG. The types that you can specify are:

ES
Specifies a VSAM entry-sequenced data set.

KS
Specifies a VSAM key-sequenced data set.

LS
Specifies a VSAM linear space data set.

RR
Specifies a VSAM relative record data set.

If you are using DATACLAS in place of RECORG, explicitly specify valid LRECL and KEYLEN values for a VSAM key-sequenced data set (RECORG(KS)).

If you do not specify RECORG, SMS assumes a physical sequential (PS) or partitioned (PO) data set.

RECORG is mutually exclusive with RECFM.

Note: Without SMS, the system syntax checks and then ignores the RECORG operand.

PROTECT
Specifies the DASD data set or the first data set on a tape volume is to be RACF-protected.

- For a new permanent DASD data set, the specified status must be NEW or MOD, treated as NEW, and the disposition must be either KEEP, CATALOG, or UNCATALOG. With SMS, SECMODEL overrides PROTECT.
- For a tape volume, the tape must have an SL, SUL, AL, AUL, or NSL label. The file sequence number and volume sequence number must be one (except for NSL), and PRIVATE must be assigned as the tape volume use attribute.

The PROTECT operand is not valid if a data set name is not specified or if the FCB operand or status other than NEW or MOD is specified.

COPIES((*number*)[,*group_value*])
Specifies the total number of copies of the data set to be printed, with an optional specification on the IBM 3800 printer as to how those copies can be grouped. Number is a required operand. The number

of copies that can be requested is subject to an installation limit. You can specify up to 8 group values. For more information, see *z/OS MVS JCL Reference*.

- Do not specify the COPIES operand with the DATASET operand.
- SYSOUT is the only valid data set status that you can specify with the COPIES operand.

BURST | NOBURST
Specifies a request for the burster-trimmer-stacker on IBM 3800 or 3900 output. SYSOUT is the only valid data set status that you can specify with the BURST operand.

CHARS(*table_name*)
Specifies a request for name or names of character arrangement tables (fonts) for printing a data set with the IBM 3800 or 3900 printer. You can specify up to 4 table names. The choice of fonts available is determined by your installation at system generation time. SYSOUT is the only valid data set status that you can specify with the CHARS operand.

FLASH(*overlay_name*[,*copies*])
Specifies the name of a forms overlay, which can be used by the IBM 3800 or 3900 Printing Subsystem. The overlay is "flashed" on a form or other printed information over each page of output. The forms *overlay_name* must be 1 to 4 alphabetic, numeric, or special characters (#, $, or @). Optionally, you can specify the number of *copies* on which the overlay is to be printed. The count can range from 0 to 255. To flash no copies, specify a count of zero. SYSOUT is the only valid data set status that you can specify with the FLASH operand.

MODIFY(*module_name*[:*trc*])

Specifies the name of a copy modification module, which is loaded into the IBM 3800 or 3900 Printing Subsystem. This module contains predefined data such as legends, column headers, or blanks, and specifies where and on which copies the data is to be printed. The IEBIMAGE utility program is used to define and store the module in SYS1.IMAGELIB. The *module_name* can contain 1 to 4 alphanumeric or special characters (#, $, or @.)

MODIFY is used with FLASH so that individual pages can be tailored with the MODIFY operand from the basic form of pages created by the FLASH operand.

The table reference character (*trc*) corresponds to the character set(s) specified on the CHARS operand. Values are 0 for the first table-name, 1 for the second, 2 for the third, or 3 for the fourth. If *trc* is not specified, a default character set is used. If *trc* is used, CHARS must also be specified.

SYSOUT is the only valid data set status that you can specify with the MODIFY operand.

FCB(*image_id*[| VERIFY | ALIGN])
specifies a forms control buffer (FCB) that is used to store vertical formatting information for printing, each position corresponding to a line on the form. The buffer determines the operations of the printer. It specifies the forms control image to be used to print an output data set on an IBM 3800 printer or 3211 printer. The FCB also specifies the data protection image to be used for the IBM 3525 card punch. The FCB operand is ignored for SYSOUT data sets on the 3525 card punch.

For further information about the forms control buffer, see *z/OS DFSMSdfp Advanced Services*, *Programming Support for the IBM 3505 Card Reader and IBM 3525 Card Punch* or *IBM 3800 Printing Subsystem Programmer's Guide*.

image_id
Specifies 1-to-4 alphanumeric or the special characters #, $, or @ that identify the image to be loaded into the forms control buffer (FCB).

- For a 3211 printer, IBM provides two standard FCB images, STD1 and STD2. STD1 specifies that 6 lines per inch are to be printed on an 8.5 inch form. STD2 specifies that 6 lines per inch are to be printed on an 11 inch form.
- For a 3800 Printing Subsystem, IBM provides another standard FCB image, STD3, which specifies output of 80 lines per page at 8 lines per inch on 11-inch long paper.

STD1 and STD2 (standard FCB images) should not be used as image_ids for the SYSOUT data set unless established by your installation at system generation time.

If the image_id information is incorrectly coded, the default for the 3211 printer is the image currently in the buffer. If there is no image in the buffer, the operator is requested to specify an image. For the 3800 printer, the machine default is 6 lines per inch for any size form that is on the printer.

ALIGN
Specifies the operator should check the alignment of the printer forms before the data set is printed. The ALIGN subparameter is ignored for SYSOUT data sets and is not used by the 3800 printer.

VERIFY
Specifies the operator should verify that the image displayed on the printer is the wanted one. The VERIFY subparameter is ignored for SYSOUT data sets.

FORMS(*forms_name*)
Specifies the name of the form on which the output from the SYSOUT data set is to be printed. Specify 1-to-4 alphanumeric or the special characters #, $, or @ for the forms name. SYSOUT is the only valid data set status that you can specify with the FORMS operand.

OUTDES(*output_descriptor_name*{,...})
Specifies a list of installation-defined output descriptors that were created by OUTPUT JCL statements in the LOGON procedure or by the TSO/E OUTDES command. Specifying the OUTDES operand eliminates the need to supply information related to the printer or the type of printing to be done.

You can specify up to 128 output descriptors associated with the SYSOUT data set. Specify 1-to-8 alphanumeric characters for the output descriptor name. The first character must be alphabetic or one of the special characters #, $, or @. SYSOUT is the only valid data set status that you can specify with the OUTDES operand.

For information about how to create output descriptors using OUTPUT JCL statements in the LOGON procedure, see *z/OS TSO/E Customization*. See "OUTDES command" on page 188 for information about using the TSO/E OUTDES command to dynamically create output descriptors.

SPIN(UNALLOC | NO)
Specifies when the system should make the SYSOUT data set available for printing.

UNALLOC
Specifies that the system should make the SYSOUT data set available for printing immediately after deallocation.

NO
Specifies that the system should make the SYSOUT data set available for printing when you log off or at the end of the batch job.

If the SPIN keyword is not specified, ALLOCATE assumes SPIN=UNALLOC.

When the SPIN keyword is specified, you must also specify UNALLOC or NO. If you specify a parameter that is not UNALLOC or NO, or the parameter is missing, ALLOCATE will prompt you to specify the parameter.

The SPIN keyword specified on the FREE command overrides the SPIN keyword specified on the ALLOCATE command.

If the SEGMENT keyword is specified on the ALLOCATE command, the system prints the SYSOUT data set regardless of the SPIN specification on either the ALLOCATE command or FREE command.

SEGMENT(*page_count*)
Specifies the number of pages that are written to the SYSOUT data set before spin-off processing begins. SEGMENT can be a number, 1-99999. You can use SEGMENT to allow part of a job's output to be printed while the job is still running, or to allow multiple segments of a job's output to print simultaneously on multiple printers. See *z/OS MVS JCL Reference*, for more information about the SEGMENT keyword.

DSNTYPE(LIBRARY{,1|,2}| PDS | HFS | PIPE |LARGE | BASIC | EXTREQ | EXTPREF)
Specifies the type of data set to be allocated.

LIBRARY
Specifies a partitioned data set extended (PDSE). LIBRARY uses the PDSE_VERSION parameter in IGDSMS*xx* or its default to determine which version of PDSE to allocate.

LIBRARY,1
Specifies a version 1 partitioned data set extended (PDSE).

LIBRARY,2
Specifies a version 2 partitioned data set extended (PDSE). Version 2 offers more efficient directory usage.

PDS
Specifies a partitioned data set (PDS).

HFS
Specifies a UNIX file system. For better performance, do not use this type of data set. Instead, define a VSAM linear data set and define a z/OS file system (zFS) in it.

PIPE
Specifies a first-in first-out (FIFO) special file, which is also called a named pipe. If you specify PIPE, you must also specify PATH and not specify DATASET or DSNAME.

LARGE
Specifies a large format sequential data set. It can have a size greater than 65535 tracks on a single volume.

BASIC
Specifies a basic format sequential data set. It is limited to no more than 65535 tracks per volume, which is about 3.6 GB.

EXTREQ
Specifies that the data set must be extended format. It can be sequential or VSAM. It can be striped, compressed format or neither.

EXTPREF
Specifies that the data set should be allocated as extended format, if possible. If not possible, allocate the data set as basic format.

If you omit DSNTYPE, the type of data set is determined by other data set attributes, the data class for the data set, or an installation default.

UCS(*universal_character_set_name*)
Specifies the universal character set name or font name to be used when printing SYSOUT data sets. The UCS name can contain up to 4 alphanumeric characters. If you do not specify the CHARS operand, the system uses the UCS operand as the default. SYSOUT is the only valid data set status that you can specify with the UCS operand unless the UNIT operand specified a directly allocated printer, not a JES-printer.

WRITER(*external_writer_name*)
Specifies a name for use in processing or selecting a SYSOUT data set. If you specify the external writer name, the system uses it instead of JES2 or JES3. The writer name can contain 1 to 8 alphanumeric or special characters #, $, or @. SYSOUT is the only valid data set status that you can specify with the WRITER operand.

A common use of this parameter is to specify the name of an external writer routine to be used to pass JCL to JES2/JES3. For example:

```
WRITER(INTRDR)
```

PATH(*pathname*)
Identifies a UNIX file.

A path name consists of the names of the directories from the root to the file being identified, and the name of the file. The form is /name1/name2/.../name*n*.

A path name begins with a slash (/). The system treats any consecutive slashes like a single slash. The path name can be 2 to 250 characters, including the initial slash.

Value for a path name consists of printable characters from X'40' to X'FE'. A file name can contain characters outside this range but these characters cannot be specified in the JCL. Enclose the path name in apostrophes if it contains any character other than the following characters:

```
Uppercase letters                Numbers
Special characters (#,$, or @)   Slash (/)
Asterisk (*)                     Plus (+)
Hyphen (-)                       Period (.)
Ampersand (&)
```

A path name is case-sensitive. Thus, '/usr/joe' and /usr/JOE define two different files.

If you specify either OCREAT alone, or OCREAT and OEXCL, on the PATHOPTS operand and if the file does not exist, MVS performs an `open()` function. The options from PATHOPTS, the path name from the PATH operand, and the options from PATHMODE (if specified) are used in the `open()`. MVS uses the `close()` function to close the file before the application program receives control.

For status group options other than OCREAT and OEXCL, the description in this documentation assumes that the application or OPEN macro passes the operands to the `open()` function without modification. That is, this application uses dynamic allocation information retrieval (the DYNALLOC macro) to retrieve the subparameters specified for PATHOPTS and passes the subparameters to the `open()` function or the OPEN macro does the equivalent. The application program can ignore or modify the information specified in the JCL or on the ALLOCATE command.

When the PATH operand is specified on the ALLOCATE command, you can specify only the following operands:

- BLKSIZE
- BUFNO
- DSNTYPE
- DUMMY
- FILEDATA
- LRECL
- NCP
- PATHDISP
- PATHMODE
- PATHOPTS
- RECFM
- REUSE
- TERM

Note: Allocation verifies the validity of the path name. However, there is no ENQ or locking of the path name, so it is possible to modify a path name component, even in an asynchronous process. Doing this might cause errors in OPEN or unexpected results with no errors reported.

Note: For programs that use a statement with the PATH keyword, do one of the following tasks:

- Use dynamic allocation information retrieval to obtain the information specified for PATH, PATHOPTS and PATHMODE, and pass it to the `open()` callable service. See *z/OS UNIX System Services Programming: Assembler Callable Services Reference*, for details for using `open()`.
- Use the C/370™ `fopen(//dd: )` function. `fopen()` handles the differences between DD statements with PATH and DSN specified. See *z/OS XL C/C++ Runtime Library Reference* for details on using `fopen()`.
- Use the OPEN macro for BSAM, QSAM, or VSAM as described in *z/OS DFSMS Using Data Sets*.

PATHDISP(*[normal_disposition] [, abnormal_disposition]*)
Specifies the disposition of a UNIX file upon normal and abnormal (conditional) TSO/E session termination.

normal_disposition
Indicates the disposition of the UNIX file upon normal TSO/E session termination. Valid values are:

KEEP
Specifies that the file should be kept.

DELETE
Specifies that the file should be deleted.

abnormal_disposition
Indicates the disposition of the UNIX System Services file upon abnormal (conditional) TSO/E session termination. Valid values are:

KEEP
Specifies that the file should be kept.

DELETE
specifies that the file should be deleted.

PATHMODE(*file_access_attribute*)
Specifies the file access attributes when the PATHOPTS operand also specifies OCREAT.

If you specify either OCREAT alone, or OCREAT and OEXCL, on the PATHOPTS operand, and if the file does not exist, then MVS performs an `open()` function. The options from PATHOPTS, the path name from the PATH operand, and the options from PATHMODE (if specified) are used in the `open()`. MVS uses the `close()` function to close the file before the application program receives control.

For status group options other than OCREAT and OEXCL, the description in this documentation assumes that the application or OPEN macro passes the operands to the `open()` function without modification. That is, this application or OPEN macro uses dynamic allocation information retrieval (the DYNALLOC macro) to retrieve the subparameters specified for PATHOPTS and passes the subparameters to the `open()` function. The application program can ignore or modify the information specified in the JCL or on the ALLOCATE command.

You can specify up to 14 file access attributes; separate each with a comma. The system treats duplicate specifications as a single specification.

Subparameter Definition:

SIRUSR
Specifies permission for the file owner to read the file.

SIWUSR
Specifies permission for the file owner to write the file.

SIXUSR
Specifies permission for the file owner to search, if the file is a directory, or to execute, for any other file.

SIRWXU
Specifies permission for the file owner to read, write, and search, if the file is a directory, or to read, write, and execute, for any other file.

This value is the bit inclusive OR of SIRUSR, SIWUSR, and SIXUSR.

SIRGRP
Specifies permission for users in the file group class to read the file.

SIWGRP
Specifies permission for users in the file group class to write the file.

SIXGRP
Specifies permission for users in the file group class to search, if the file is a directory, or to execute, for any other file.

SIRWXG
Specifies permission for users in the file group class to read, write, and search, if the file is a directory, or to read, write, and execute, for any other file.

This value is the bit inclusive OR of SIRGRP, SIWGRP, and SIXGRP.

SIROTH
Specifies permission for the users in the file other class to read the file.

SIWOTH
Specifies permission for users in the file other class to write the file.

SIXOTH
Specifies permission for users in the file other class to search, if the file is a directory, or to execute, for any other file.

SIRWXO
Specifies permission for users in the file other class to read, write, and search, if the file is a directory, or to read, write, and execute, for any other file.

This value is the bit inclusive OR of SIROTH, SIWOTH, and SIXOTH.

SISUID
Specifies that the system set the user ID of the process to be the same as the user ID of the file owner when the file is run as a program.

SISGID
Specifies that the system set the group ID of the process to be the same as the group ID of the file owner when the file is run as a program. The group ID is taken from the directory in which the file resides.

When creating a new UNIX file, if you do not code a PATHMODE operand on a DD statement with a PATH operand, the system sets the permissions to zero, which prevents access by all users. If the UNIX file already exists, PATHMODE is checked for syntax but ignored. The permission bits are left as they are set.

PATHOPTS(*file_options*)
Specifies the file access and status used when accessing a file specified on the PATH operand. You can specify up to 7 file options; separate each with a comma. The system treats duplicate specifications as a single specification.

```
Access Group:        Status Group:
------------         ------------
(choose only 1)      (choose up to 6)
ORDONLY              OAPPEND
OWRONLY              OCREAT
ORDWR                OEXCL
                     ONOCTTY
                     ONONBLOCK
                     OTRUNC
```

Note: If you specify more than one Access Group, the system ignores them and uses ORDWR.

If you specify either OCREAT alone, or OCREAT and OEXCL, on the PATHOPTS operand, and if the file does not exist, then MVS performs an `open()` function. The options from PATHOPTS, the path name from the PATH operand, and the options from PATHMODE (if specified) are used in the `open()`. MVS uses the `close()` function to close the file before the application program receives control.

For status group options other than OCREAT and OEXCL, the description in this documentation assumes that the application or OPEN macro passes the operands to the `open()` function without modification. That is, this application or OPEN macro uses dynamic allocation information retrieval (the DYNALLOC macro) to retrieve the subparameters specified for PATHOPTS and passes the subparameters to the `open()` function. The application program can ignore or modify the information specified in the JCL or on the ALLOCATE command.

Sub-parameter definition:

ORDONLY
Specifies this access group so that the program can open the file for reading.

OWRONLY
Specifies that the program can open the file for writing.

ORDWR
Specifies that the program can open the file for reading and writing. Do not use this option for a FIFO special file.

OAPPEND
Specifies that the system sets the file offset to the end of the file before each write, so that data is written at the end of the existing file.

OCREAT
Specifies that the system is to create the file. If the file already exists, the operation fails if OEXCL is specified, and opens the existing file if OEXCL is not specified.

OEXCL
Specifies that, if the file does not exist, the system is to create it. If the file already exists, open() fails. Note that the system ignores OEXCL if OCREAT is not also specified.

ONOCTTY
Specifies that, if the PATH operand identifies a terminal device, open() does not also make the terminal device the controlling terminal of the process and the session.

ONONBLOCK
Specifies the following, depending on the type of file:

- For a FIFO special file with ORDONLY option set:

  ONONBLOCK specifies read-only opening of the file. If ONONBLOCK is not specified, the read-only open() blocks until a process opens the file for writing.

- For a FIFO special file with OWRONLY option set:

  ONONBLOCK specifies that the system immediately process a request for a write-only open() of the file, if a process has already opened the file for reading. If the file is not open for reading, the system returns an error. If ONONBLOCK is not specified, the write-only open() blocks until a process opens the file for reading.

- For a character special file that supports a nonblocking open():

  ONONBLOCK specifies that the system immediately returns if it cannot open a file because the device is not ready or available. If ONONBLOCK is not specified, the open() blocks until the device is ready or available.

  Specifications of ONONBLOCK have no effect on other file types.

OSYNC
Specifies that the system is to move data from buffer storage to permanent storage before returning control from a callable service that performs a write.

OTRUNC
Specifies that the system is to truncate the file to zero length if all of the following conditions are true:

- The file specified on the PATH operand exists.
- The file is a regular file.
- The file successfully opened with ORDWR or OWRONLY.

The system does not change the mode and owner. OTRUNC has no effect on FIFO special files or directories.

FILEDATA(BINARY | TEXT | RECORD)
Controls the data conversion method, performed by the network file system client, when accessing network files on a different system. For information of using UNIX file system such as NFS, see *z/OS DFSMS Using Data Sets*. The other system might be OS/390®, AIX®, or certain other kinds of systems. The FILEDATA keyword is used to describe the content type of a z/OS UNIX file so that the system can determine how to process the file.

BINARY
Specifies that the file described by the DD statement is a byte-stream file and does not contain record delimiters. The access method does not insert or delete record delimiters.

If you do not code the FILEDATA operand, the system assigns a default value of BINARY to the UNIX file.

TEXT
Specifies that the file described by the DD statement contains records delimited by the EBCDIC newline character (x'15').

See the appropriate DFSMS publications for more details about the Network file system client and its conversion methods.

You need to code the PATH operand together with the FILEDATA operand.

You can code the FILEDATA operand together with the following ALLOCATE operands: BLKSIZE, BUFNO, DSNTYPE, DUMMY, LRECL, NCP, PATHDISP, PATHMODE, PATHOPTS, RECFM.

RECORD
Indicates that the data consists of records with prefixes. The record prefix contains the length of the record that follows. On output, the access method inserts a record prefix at the beginning of each record. On input, the access method uses the record prefix to determine the length of each record. The access method does not return the prefix as part of the record. Code FILEDATA(RECORD) when you cannot code FILEDATA(TEXT) because your data might contain bytes that are considered delimiters.

Note: The record prefix for FILEDATA(RECORD) is mapped by the IGGRPFX macro. This is different from the record descriptor word (RDW) that is in z/OS physical sequential format-V data sets.

RLS(NRI | CR)
Specifies the level of record sharing, or *sharing protocol*, for a VSAM data set in a sysplex. See *z/OS DFSMS Using Data Sets*, for a description of sharing protocols and to determine whether your application can run in a sharing environment without modification.

NRI
Specifies no read integrity (NRI). An application can read uncommitted changes to a data set made by another application.

CR
Specifies consistent read (CR). An application can read only committed changes to a data set made by another application. An application might require changes if it attempts to read changes to a data set that was allocated with a specification of CR.

Do not use any of the following ALLOCATE operands with RLS: BURST, CHARS, COPIES, DDNAME, DSNTYPE, FLASH, MODIFY, OUTPUT, PATH, PATHOPTS, PATHMODE, PATHDISP, SEGMENT, SPIN, SYSOUT, UCS.

EATTR(NO | OPT)
Specifies whether the data set can support extended attributes (format 8 and 9 DSCBs) or not. To create such data sets, you can include extended address volumes (EAVs) in specific storage groups or specify an EAV on the request or direct the Allocation to an esoteric containing EAV devices.

By definition, a data set with extended attributes can reside in the extended address space (EAS) on an extended address volume (EAV). This parameter can be specified for non-VSAM data sets and for VSAM data sets. If EATTR is not specified, VSAM data sets can have format 8 and 9 DSCBs by default, while non-VSAM data sets cannot.

The EATTR value has no effect for DISP OLD processing, even for programs that might open a data set for OUTPUT, INOUT, or OUTIN processing. The value on the EATTR parameter is used for requests when the data set is newly created.

NO
Specifies that no extended attributes are available. The data set cannot have extended attributes (format 8 and 9 DSCBs) or reside in EAS.

OPT
Specifies that extended attributes are optional. The data set can have extended attributes and reside in EAS.

ALLOCATE command return codes

Table 4 on page 41 lists the return codes of ALLOCATE command.

Table 4: ALLOCATE command return codes

Return code	Meaning
0	Allocation successful.
12	Allocation unsuccessful. An error message has been issued.

ALLOCATE command examples

Example 1: Allocate your terminal as a temporary input data set

```
allocate da(*) file(ft01f001)
```

Example 2: Allocate an existing cataloged data set

Known:

- The name of the data set: MOSER7.INPUT.DATA.

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

Note that you do not have to specify the user ID, MOSER7, as an explicit qualifier.

Example 3: Allocate an existing data set that is not cataloged

Known:

- The data set name: SYS1.PTIMAC.AM
- The volume serial number: B99RS2
- The ddname: SYSLIB

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

Example 4: Allocate a new data set with the attributes of an existing model data set

Known:

- The name that you want to give the new data set: MOSER7.NEW.DATA
- The name of the model data set: MOSER7.MODEL.DATA

```
alloc da(new.data) like(model.data)
```

Example 5: Allocate a new data set that differs from an existing model data set only in its space allocation

Known:

- The name that you want to give the new data set: MOSER7.NEW2.DATA
- The name of the model data set: MOSER7.MODEL.DATA

- The desired space attributes for the new data set: primary 10 tracks, secondary 5 tracks

```
alloc da(new2.data) space(10,5) tracks like(model.data)
```

Example 6: Allocate a new sequential data set with space allocated in tracks

Known:

- The new data set name: MOSER7.EX1.DATA
- The number of tracks: 2
- The logical record length: 80
- The DCB block size: 8000
- The record format: fixed block

```
alloc da(ex1.data) dsorg(ps) space(2,0) tracks lrecl(80) +
blksize(8000) recfm(f,b) new
```

Example 7: Allocate a new partitioned data set with space allocated in blocks

Known:

- The new data set name: MOSER7.EX2.DATA
- The block length: 200 bytes
- The logical record length: 100
- The DCB block size: 200
- The number of directory blocks: 2
- The record format: fixed block

```
alloc da(ex2.data) dsorg(po) block(200) space(10,10) +
lrecl(100) blksize(200) dir(2) recfm(f,b) new
```

Example 8: Allocate a new sequential data set with default space quantities

Known:

- The new data set name: MOSER7.EX3.DATA
- The block length: 800 bytes
- The logical record length: 80
- The record format: fixed block

```
alloc da(ex3.data) block(800) lrecl(80) dsorg(ps) +
recfm(f,b) new
```

Example 9: Allocate a new sequential data set using an attribute list

Known:

- The name that you want to give the new data set: MOSER7.EX4.DATA
- The number of tracks expected to be used: 10
- DCB operands are in an attribute list named: ATRLST1

```
attrib atrlst1 dsorg(ps) lrecl(80) blksize(3200)

alloc da(ex4.data) new space(10,2) tracks using(atrlst1)
```

Example 10: Allocate a new sequential data set with space allocated in blocks and using an attribute list

Known:

- The new data set name: MOSER7.EX5.DATA
- The block length: 1000 bytes
- The DCB attributes taken from attribute list: ATRLST3

```
attrib atrlst3 dsorg(ps) lrecl(80) blksize(3200)

alloc da(ex5.data) using(atrlst3) block(1000) +
space(20,10) new
```

Example 11: Allocate a new sequential data set with default space quantities and using an attribute list

Known:

- The new data set name: MOSER7.EX6.DATA
- The DCB attributes taken from attribute list: ATRLST5

```
attrib atrlst5 dsorg(ps) lrecl(80) blksize(3200)

alloc da(ex6.data) using(atrlst5) new
```

Example 12: Allocate a new data set to contain the output from a program

Known:

- The data set name: MOSER7.OUT.DATA
- The ddname: OUTPUT
- You do not want to hold unused space.

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

Example 13: Allocate an existing multi-volume data set to SYSDA, with one device mounted for each volume

Known:

- The data set name: MOSER7.MULTIVOL.DATA
- Volumes: D95VL1, D95VL2, D95VL3
- The ddname: SYSLIB

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

Example 14: Allocate an existing data set as the second file of a standard-label tape

Known:

- The data set name: MOSER7.TAPE1.DATA
- The volume: TAPEVL
- The unit: 3480

```
alloc dataset('moser7.tape1.data') label(sl) +
unit(3480) volume(tapevl) position(2)
```

Example 15: Allocate an output data set using the FCB and COPIES operands to request formatted copies of an output data set

Known:

- The ddname: OUTPUT

- The FCB image desired: STD1
- The number of copies: 10

```
alloc file(output) sysout fcb(std1) copies(10)
```

Example 16: Allocate a new tape data set using the PROTECT operand to request RACF protection

Known:

- The data set name: MOSER7.TAPE2.DATA
- The volume: TAPEV2
- The unit: 3490

```
alloc da(tape2.data) unit(3490) label(sl) position(1) +
volume(tapev2) protect new
```

Example 17: Allocate a new DASD data set using the PROTECT operand to request RACF protection

Known:

- The data set name: MOSER7.DISK.DATA
- The logical record length: 80
- The DCB block size: 8000
- The record format: fixed block
- The number of tracks: 2

```
alloc da(disk.data) dsorg(ps) space(2,0) tracks +
lrecl(80) blksize(8000) recfm(f,b) protect new
```

Example 18: Concatenate some data sets

Known:

- The data set names: A.CLIST, B.CLIST, C.CLIST
- The ddname: SYSPROC

```
alloc file(sysproc) dataset(a.clist,b.clist,c.clist) +
shr reuse
```

You cannot directly add another data set to a concatenation. There are two ways to add another data set to a data set concatenation:

1. Use the FREE command to deallocate or free the data sets in the concatenation. Then reallocate the entire concatenation, including the data set to be added, using the ALLOCATE command.
2. Specify the REUSE operand with the ALLOCATE command when you concatenate. The REUSE operand specifies the file name being allocated is to be freed and reallocated if it is currently in use.

Example 19: Allocate a data set defined by a DD statement as a SYSOUT to a specific print form

Known:

- The ddname: PAYROLL
- The output descriptor: PRINTER1
- The print form name: CHEK

```
alloc f(payroll) sysout forms(chek) outdes(printer1)
```

Example 20: Allocate a SYSOUT data set specifying the member name of an installation program that writes the data set

Known:

- The ddname: REPORTA
- The writer name: OURWRIT
- The output descriptor: DESCRIPT

```
alloc f(reporta) sysout writer(ourwrit) outdes(descript)
```

Example 21: Allocate a SYSOUT data set to be printed in a specific character set or print font

Known:

- The ddname: REPORTB
- The character set: GOTH
- The output descriptor: DESCRIPT

```
alloc f(reportb) sysout ucs(goth) outdes(descript)
```

Example 22: Allocate a SYSOUT data set to make it available for printing immediately after deallocation

Known:

- The name of the file: SYSPRINT

```
alloc f(sysprint) sysout spin(unalloc)
```

Example 23: Allocate a SYSOUT data set specifying the number of pages to print

Known:

- The name of the file: SYSPRINT
- Desired segment size: 500

```
alloc f(sysprint) sysout da(*) segment(500)
```

Example 24: Allocate a SYSOUT data set to be routed to a user at a remote destination

Known:

- The ddname: FREEDOM
- The destination: NEWYORK
- The user ID: LIBERTY

```
alloc f(freedom) sysout dest(newyork.liberty)
```

Example 25: Allocate an OBJECT PDS with a data class of OBJ

The following example assumes that the Storage Management Subsystem (SMS) is installed and is active.

Known:

- The data set name: SMS.PDS.OBJ
- The data class: OBJ
- The storage class: STANDARD
- The management class: TSO

- The data class attributes: LRECL (80), RECFM (FB), primary quantity (10), secondary quantity (10), directory blocks (5), AVGSIZE (800)

```
alloc da('sms.pds.obj') new dataclas(obj) storclas(standard)+
mgmtclas(tso)
```

Example 26: Override the data class, storage class, management class operands by specifying them on the ALLOCATE command

The following example assumes that SMS is installed and is active.

Known:

- The data set name: SMS.NEW.OBJ
- The data class attributes: LRECL (80), RECFM (FB), primary quantity (10), secondary quantity (10), directory blocks (5), AVGREC (U), AVGSIZE (6160)

```
alloc da('sms.new.obj') dataclas(pds) storclas(general) +
mgmtclas(temp) new
```

Example 27: Allocate three data sets using the REUSE operand

Known:

- Data set name: MY.DATA.SET
- MY.DATA.SET on volume STOR03 is cataloged
- MY.DATA.SET on volume STOR99 is uncataloged

1. Allocate MY.DATA.SET on volume STOR03:

   ```
   alloc file(x) da(my.data.set') reuse shr
   ```

2. Allocate MY.DATA.SET on volume STOR99. The REUSE operand frees the *file x* allocation for MY.DATA.SET on volume STOR03 and reallocates file *x* to MY.DATA.SET on volume STOR99.

   ```
   alloc file(x) da('my.data.set') reuse shr vol(stor99)
   ```

3. Allocate MY.DATA.SET on volume STOR99:

   ```
   alloc file(x) da('my.data.set') reuse shr
   ```

This is satisfied by the current allocation which is MY.DATA.SET on volume STOR99.

Note: REUSE does not free *file x* and then reallocate *file x* with MY.DATA.SET on volume STOR03 because it is satisfied by an existing allocation.

If you want to allocate MY.DATA.SET on volume STOR03, you can either free `file x` and then issue the same allocate command or specify volume STOR03 on the allocation.

Example 28: Allocate a new DASD data set using a system set block size and request space in a quantity of logical records

The following example assumes that Storage Management Subsystem is installed and active because of AVGREC. The system-determined block size does not require SMS.

Release overallocated space down to a track boundary when the data set is closed.

Known:

- The data set name: K9345P.REPORT2
- The logical record length: 133
- The record format: Fixed block ANSI

- The number of logical records: Primary quantity 5000, secondary quantity 500

```
alloc ds('k9345p.report2') new dsorg(ps) recfm(f,b,a) +
lrecl(133) avblock(133) avgrec(u) space(5000,500) release
```

Example 29: Allocate an output file, creating it if it does not exist

Known:

- The ddname: OUTPUT
- The pathname: **/u/userid/file.dbp**
- The disposition: Keep under all circumstances.
- Permissions: Read, write, and execute for the user; no other permissions.

```
alloc path('/u/userid/file.dbp')    +
      pathdisp(keep,keep)           +
      pathopts(owronly,ocreat)      +
      pathmode(sirwxu)              +
      file(output)
```

ALTLIB command

Use the ALTLIB command to:

- Define alternative application-level libraries of REXX execs or CLISTs.
- Indicate that user-, application-, and system-level libraries of REXX execs and CLISTs are being searched.
- Exclude one or more library levels (user, application, system) from being searched.
- Reset the search order to the system level.
- Obtain a display of the search order that is in effect.

TSO/E searches the user-, application-, and system-level libraries for REXX execs or CLISTs that are executed implicitly or when searching for REXX external functions or subroutines. For more information about implicitly executing execs and CLISTs, see "EXEC command" on page 120. For more information about REXX external functions, see *z/OS TSO/E REXX Reference*.

Search order for libraries

Table 5 on page 47 lists the search order of the user-, application-, and system-level libraries. Also shown are the ddnames associated with each library level. These ddnames can be allocated either dynamically by the ALLOCATE command or included as part of a logon procedure.

Table 5: Library search order

Search order	Library level		Associated ddname
1.	User	REXX exec	SYSUEXEC
2.	User	CLIST	SYSUPROC
3.	Application	REXX exec	Define with FILE or DATASET operand
4.	Application	CLIST	Define with FILE or DATASET operand
5.	System	REXX exec	SYSEXEC (installation can define this ddname)
6.	System	CLIST	SYSPROC

With the defaults that TSO/E provides, and before an ALTLIB command is invoked, TSO/E searches the system EXEC library (default ddname SYSEXEC) first, followed by the system CLIST library (ddname SYSPROC). Note that your system programmer can change this by:

- Defining an alternate ddname of SYSEXEC
- Indicating that TSO/E is not to search the system-level exec ddname of SYSEXEC. Then only the system-level CLIST (SYSPROC) is searched.

You can alter the default library search order by using either the ALTLIB command or the EXECUTIL command.

- Use EXECUTIL to indicate that the system-level exec ddname is to be searched for the duration of the current REXX language processor environment.
- Use ALTLIB to indicate that the system-level exec ddname is to be searched for the duration of the current application. ALTLIB always overrides EXECUTIL within an application.

Use ALTLIB DISPLAY to see which libraries are being searched for.

Using ALTLIB with most applications

With most applications, the ALTLIB command is in effect from the time that the command is entered until either another ALTLIB command is entered or the TSO/E session is ended.

Examples of applications where the ALTLIB command remains in effect for the duration of the session include TSO/E line mode, TMP READY mode, and TSO/E commands that accept subcommands, such as IPCS. This does not, however, apply to ISPF, ISPF dialogs, and similar programs.

Using ALTLIB with concurrent applications

TSO/E permits applications that allow users to perform multiple tasks to distinguish between the set of procedure libraries required to support one task and the set of procedure libraries required to support a different task.

For example, a user can edit a memo using the ISPF/PDF editor from one logical screen and interleave that task with the browsing of a dump using the IPCS dialog from a different logical screen.

Using ALTLIB in ISPF

When you use ALTLIB when ISPF is active, you can define the libraries (user, application, and system) that are active for each application. Libraries that you define while running an application are in effect only while that application has control. When the application completes, the previous libraries (user, application, and system) are automatically reactivated.

If you are in split-screen mode in ISPF and you issue the ALTLIB command from a one-screen session, the changes affect only that screen session. The ALTLIB search order is not valid across split screens.

The libraries that are originally used when an application gets control are determined through the NEWAPPL and PASSLIB parameters on the ISPF SELECT service. For more information about the SELECT service, see *z/OS ISPF Services Guide*.

When NEWAPPL is specified and PASSLIB is not specified (that is, you want to isolate the selected function from the application currently in control, but you do not want to pass library definitions specified with the ALTLIB command and ISPF LIBDEF service on to the new application), the current set of libraries, if any exist, are not used by the application being selected. The deactivation of these libraries takes place BEFORE the application is selected. The current library definitions are automatically reactivated when the application being selected terminates.

When both NEWAPPL and PASSLIB are specified (that is, you want to isolate the selected function from the application currently in control and you want to pass library definitions specified with the ALTLIB command and ISPF LIBDEF service on to the new application), the current set of libraries, if any exist, are made available to the selected application. Any changes you make to this set of libraries while this

application is running are in effect only while this application has control. After the selected application terminates, the original set of libraries is reactivated.

When NEWAPPL and PASSLIB are not specified (that is, you do not want to isolate the selected function), the current set of libraries remains in effect because the selected function does not represent a new application. If the selected function changes any of the library definitions, the changes apply through all select levels of the application of which the selected function is a part.

ALTLIB within line mode TSO/E works just like an ISPF application. However, if you use ALTLIB from within line mode TSO/E and start ISPF, the libraries you specified in line mode TSO/E will not be available until ISPF is terminated.

Using ALTLIB in the IPCS dialog

When you activate the IPCS dialog for a logical screen, the IPCS dialog establishes an ALTLIB environment with the same search order that is in effect before the first ALTLIB command is invoked. See "Search order for libraries" on page 47 for the order in which TSO/E searches the libraries. This environment is used solely for IPCS dialog processing for the logical screen.

Although the initial environments are similar, the IPCS environment maintains a separate ALTLIB environment from that of ISPF. IPCS controls separate ALTLIB environments for each ISPF logical screen in which the IPCS is invoked.

When you direct commands to the IPCS dialog, the EXEC command uses the ALTLIB environment associated with the ISPF logical screen in which the IPCS dialog is invoked. When you direct the ALTLIB command to the IPCS dialog, defining or excluding one or more libraries, only the ALTLIB environment associated with that IPCS dialog for that logical screen will change.

When you direct commands to ISPF within the IPCS dialog, the EXEC command uses the ALTLIB environment associated with the particular ISPF application that IPCS is running. When you direct the ALTLIB command to ISPF, only the ALTLIB environment associated with that ISPF application will change.

Only the IPCS dialog maintains a separate ALTLIB environment. Native IPCS does not maintain a separate ALTLIB environment.

Note: Do not use the QUIET option of ALTLIB in the IPCS dialog. IPCS does not make ISPF services available to TSO/E commands that IPCS invokes.

For more information about using the ALTLIB command when the IPCS dialog is active, see *z/OS MVS IPCS User's Guide*, and *z/OS MVS IPCS Commands*.

Stacking Application-Level library requests

Application-level REXX exec and CLIST requests can be stacked up to eight requests each. Because the application-level requests are stacked, you can activate a REXX exec or CLIST and then reissue the request for the same REXX exec or CLIST and the first request will still exist. When you stack application-level library requests for REXX execs or CLISTs, the last application level you activate becomes the current one. Only the top, or current application-level request is active.

For example, if you activate an application-level CLIST,

```
altlib activate application(clist) dataset('userid.ds1')
```

and then unconditionally activate another application-level CLIST,

```
altlib activate application(clist) dataset('otherid.ds5') uncond
```

the second request becomes current and the first request is stacked under it.

If you entered the command, ALTLIB DISPLAY, to display the search order, the display at your terminal will look similar to the following:

```
IKJ79322I   Current search order (by DDNAME) is:
IKJ79326I                Application-level CLIST DDNAME=SYS00027
IKJ79321I                                Stacked   DDNAME=SYS00026
IKJ79327I                System-level EXEC       DDNAME=SYSEXEC
IKJ79328I                System-level CLIST      DDNAME=SYSPROC
```

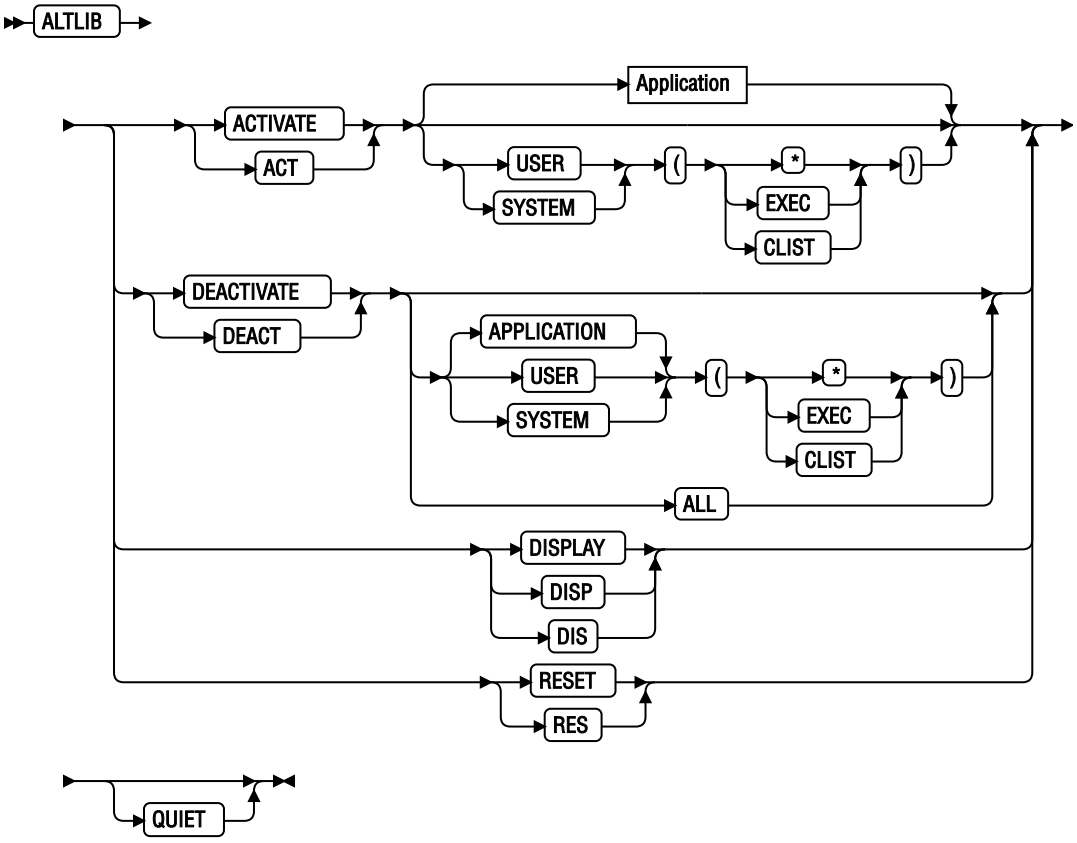
Deactivate the application-level for CLIST to remove the second request and make the first request current. Or, you can clear all requests and reset the original library search order. For example, to clear only the current request issue:

```
altlib deactivate application(clist)
```

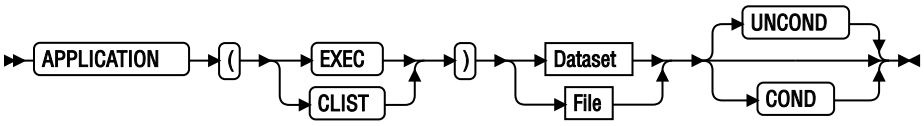
To clear all stacked application-level requests and leave the user and system levels as they are, issue:

```
altlib deactivate application(*)
```

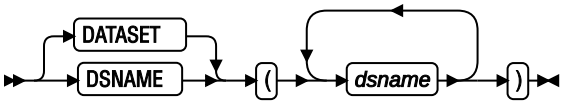
ALTLIB command syntax



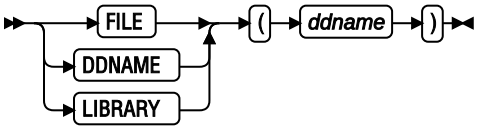
Application



Dataset



File



ALTLIB command operands

ACTIVATE | ACT
indicates that you want to include the specified library level when searching for a REXX exec or CLIST.

DEACTIVATE | DEACT | DEA
indicates that you want to exclude the specified library level when searching for a REXX exec or CLIST.

DISPLAY | DISP | DIS
requests information about the search order the EXEC command processor currently uses to find a REXX exec or CLIST.

RESET | RES
resets the libraries searched to system-level REXX execs and CLISTs only.

USER
indicates that the user-level REXX execs and CLISTs are to be activated or deactivated. User-level REXX execs and CLISTs are those data sets concatenated to the ddname SYSUPROC for both CLISTs and REXX execs and the data sets concatenated to ddname SYSUEXEC for REXX execs only.

APPLICATION
indicates that the application-level REXX execs and CLISTs are to be activated or deactivated. Application-level execs and CLISTs are those data sets defined with the DATASET or FILE operands.

SYSTEM
indicates that the system-level REXX execs and CLISTs are to be activated or deactivated. System-level execs and CLISTs are the data sets that are concatenated to the ddname SYSPROC for both REXX execs and CLISTs or those data sets that are concatenated to the ddname SYSEXEC for REXX execs only.

ALL
indicates that you want to deactivate all library levels, user, application, and system, of REXX execs and CLISTs.

(EXEC)
indicates that you want to activate or deactivate REXX execs at the level you specify (user, application, or system).

(CLIST)
indicates that you want to activate or deactivate CLISTs at the level that you specify (user, application, or system).

(*)
indicates that you want to activate or deactivate REXX execs and CLISTs at the level you specify (user, application, or system).

DATASET(*dsname*) | DSNAME(*dsname*)
specifies a data set list to define an application-level library of REXX execs or CLISTs. When specifying DATASET or DSNAME:

- The data sets must be cataloged partitioned data sets when you issue the ALTLIB command.
- The maximum number of data sets you can list is fifteen. Use the FILE operand if you want to specify more than fifteen.
- The data sets must all have the same record format (RECFM).
- If the data sets have different block sizes, you can specify them in any order of block sizes.
- Member names cannot be specified in the list of data sets.

FILE(*ddname*) | DDNAME(*ddname*) | LIBRARY(*ddname*)
specifies a ddname that defines an application-level library.

- The ddname must be allocated before issuing the ALTLIB command.
- The ddname must be allocated with the permanently allocated attribute to ensure that the system does not automatically deallocate the ddname when the allocation control limit is exceeded. Note that the data sets allocated in a LOGON procedure or by the ALLOCATE command are automatically allocated with this attribute, however, if you access dynamic allocation directly, using SVC 99, you need to specify this attribute. For more information about the permanently allocated attribute, see *z/OS MVS Programming: Authorized Assembler Services Guide*.
- To avoid errors when the EXEC command runs, specify only cataloged partitioned data sets.

UNCOND | COND

UNCOND
activates the specified application-level library even if another application-level library of the same type, CLIST or REXX exec, is active within the current application. Up to eight application-level CLIST and REXX exec requests can be stacked. (See "Stacking Application-Level library requests" on page 49 for an explanation of stacking.)

COND
activates the specified application-level library only if another application-level library of the same type, CLIST or REXX exec, is *not* active within the current application. If you issue the ALTLIB command with the COND keyword and there is already an application-level library in effect, a message is displayed and a non-zero return code is set.

QUIET
indicates that you want messages saved and not displayed at the terminal. Messages can be saved in the ISPF shared pool when QUIET is used and ISPF is active. Variable IKJADM1 contains the first message, variable IKJADM2 contains the second message, and so on. Variable IKJADM contains the number of messages returned for the invocation of ALTLIB according to these rules:

- If you specify ALTLIB with QUIET, IKJADM is reset to the number of messages returned for that invocation of ALTLIB.
- If you do not specify the QUIET operand, IKJADM is not reset. It equals the number of messages returned for the last invocation of ALTLIB with QUIET.
- QUIET takes effect after TSO/E determines that the ALTLIB command is syntactically correct. If the command is not syntactically correct, then IKJADM equals 0 and a return code of 20 is returned indicating a syntax error.

QUIET saves up to 99 messages.

IKJADM1 echoes the command entered in IKJADM1. For example,

```
IKJADM = 4
IKJADM1= ALTLIB DISPLAY QUIET
IKJADM2= IKJ79322I Current search order (by DDNAME) is:
IKJADM3= IKJ79327I System-level EXEC       DDNAME=SYSEXEC
IKJADM4= IKJ79328I System-level CLIST      DDNAME=SYSPROC
```

REXX execs and CLISTs may use the variables IKJADM and IKJADM1 - IKJADM99 as in this example:

```
/* REXX */
ADDRESS TSO "ALTLIB DISPLAY QUIET"
ADDRESS ISPEXEC "VGET (IKJADM IKJADM1 IKJADM2 IKJADM3 IKJADM4) SHARED"
SAY 'IKJADM = 'IKJADM
SAY 'IKJADM1='IKJADM1
SAY 'IKJADM2='IKJADM2
SAY 'IKJADM3='IKJADM3
SAY 'IKJADM4='IKJADM4
```

If you use a program that invokes ALTLIB with the QUIET operand, you must take the following into consideration: ALTLIB declares IKJADM as a fixed binary integer, four bytes long. IKJADM1 - 99 are character format, 251 bytes long. If QUIET is in effect and you invoke ALTLIB from a program, messages are not displayed, but they are available to the program.

Note: Do not use the QUIET option of ALTLIB in the IPCS dialog. IPCS does not make ISPF services available to TSO/E commands that IPCS invokes.

ALTLIB command return codes

Table 6 on page 53 lists all the return codes of ALTLIB command.

Table 6: ALTLIB command return codes

Return code	Meaning
0	Processing successful. Informational messages might have been issued.
4	An alternative library does not exist for this type (REXX exec or CLIST); none deactivated.
8	An application-level library already exists for this type (REXX exec or CLIST). The new application-level library was not activated. Issued only when you specify the COND parameter.
10	User- or system-level CLIST activated; User- or system-level exec cannot be activated because a REXX language processor environment has not been established. Contact your system programmer to diagnose problems with TSO/E programs IRXECUSP and IRXINIT.
16	A required ddname was not previously allocated.
20	Severe error. More information is contained in messages.

ALTLIB command examples

Example 1

Operation: Search for CLISTs in a user-level library before application- or system-level libraries. First allocate a user-level ddname, then activate the user-level CLISTs.

```
allocate fi(sysuproc) da('id.clist') shr reu

altlib activate user(clist)
```

Example 2

Operation: Display the search order currently used to find a REXX exec or CLIST.

```
altlib display
```

The output at your terminal might be similar to the following:

```
IKJ79322I  Current search order (by DDNAME) is:
IKJ79327I      System-level EXEC             DDNAME=SYSEXEC
IKJ79328I      System-level CLIST            DDNAME=SYSPROC
```

Example 3

Operation: Define an application-level CLIST library even if another application-level CLIST library exists, and request that messages are not to be displayed.

```
altlib activate application(clist) dataset(clist.name) uncond quiet
```

ATTRIB command

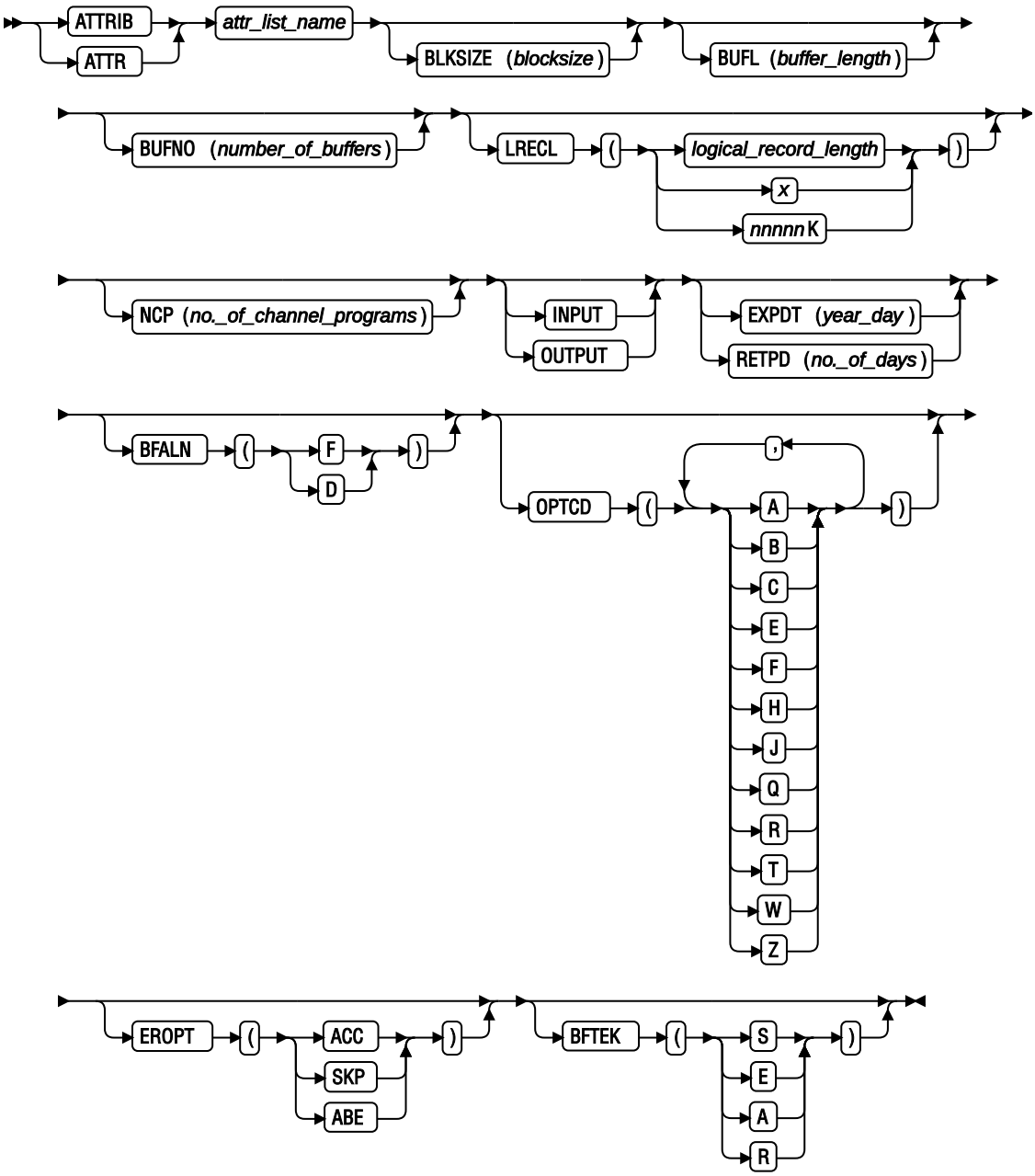
Use the ATTRIB command to build a list of attributes for non-VSAM data sets that you intend to allocate dynamically. During the remainder of your terminal session, you can have the system this list for data set attributes when you enter the ALLOCATE command. The ALLOCATE command converts the attributes into

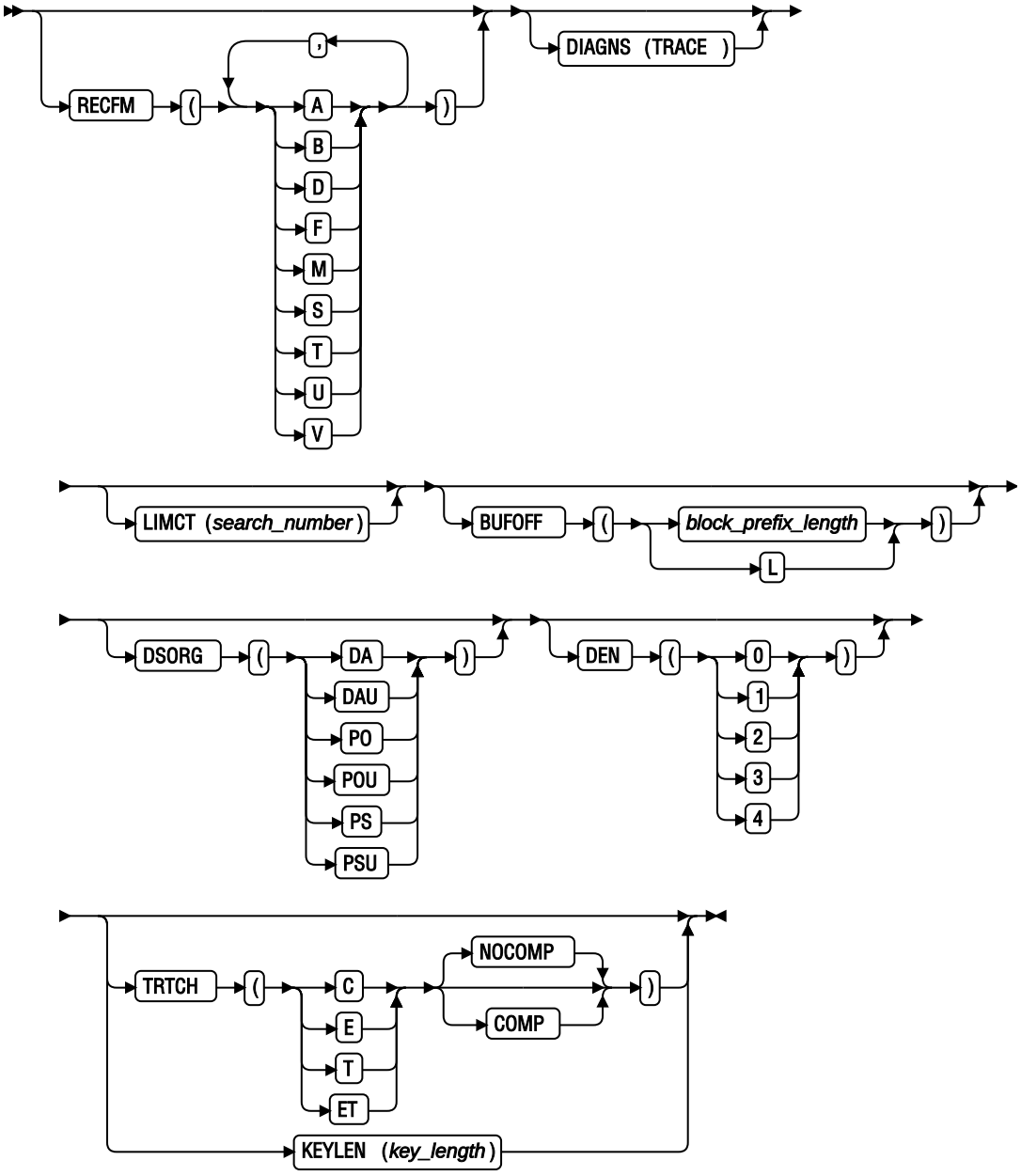
DCB operands and LABEL operands for data sets being allocated. See the subparameters of the DCB parameter in *z/OS MVS JCL Reference*.

The ATTRIB command allocates a file with the same name as your *attr_list_name*. You can use the LISTALC command with the STATUS operand 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 operand of the ALLOCATE command. Because this is a NULLFILE allocation, it is subject to use and modification by other commands. Therefore, it is advisable to allocate those data sets for which the attribute list was built before you issue any commands that might cause NULLFILE allocation, such as LINK or RUN.

With the LIKE operand and the DCB operands on the ALLOCATE command, you do not have to use the ATTRIB command.

ATTRIB command syntax





ATTRIB command operands

attr_list_name
specifies the name for the attribute list. You can specify this name later as an operand of the ALLOCATE command. The name must consist of 1 to 8 alphanumeric and the special characters #, $, or @, or both must begin with an alphabetic or special character, and must be different from all other attribute list names and ddnames that exist during 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 32760 bytes.

The block size 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. 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. 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. Because the number of logical records can vary, you must estimate the optimum block size and the average number of records for each block based on your knowledge of the application that requires the I/O.
- RECFM(U), then the block size can be any value up to what is supported by the device or 32760, whichever is smaller. If allocated to a TSO/E terminal and BLKSIZE(80) is coded, then one character (the last byte) is reserved for an attribute character.

BUFL(*buffer_length*)
specifies the length, in bytes, of each buffer in the buffer pool. Specify a decimal number for buffer_length. The number must not exceed 32760.

If you omit this operand and the system acquires buffers automatically, the BLKSIZE and KEYLEN operands are 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. Specify a decimal number for number_of_buffers. The number must not exceed 255. You might be limited to a smaller number of buffers depending on the limit established at your installation. 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 should specify BUFNO if the program was designed to use buffers obtained during OPEN. You should not specify BUFNO if the program was not designed to use buffers obtained during OPEN; otherwise those buffers will not be used.
(4) Automatically with QSAM	(4) You may omit BUFNO and accept the system default, which is five or one, except with an extended format data set. For more information see *z/OS DFSMS Using Data Sets*.

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.

If the data set contains undefined-length records, omit this operand.

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.

LRECL(NNNNNK) allows users of ISO/ANSI extended logical records and QSAM locate mode users to specify a K multiplier on the LRECL operand. NNNNN can be within 1-16,384. The K indicates that the value can be multiplied by 1024.

For variable-length spanned records (VS or VBS) processed by QSAM (locate mode) or BSAM, specify LRECL (X) when the logical record exceeds 32756 bytes.

NCP(*number_of_channel_programs*)
specifies the maximum number of READ or WRITE macro instructions allowed before a CHECK macro instruction is issued. The maximum number must not exceed 255 and must be less than 255 if the address space does not have enough virtual storage. If you want to use chained scheduling, you must specify an NCP value greater than 1. If you omit the NCP operand and the application program does not specify the MULTSDN parameter on the DCBE macro, the default value is 1. This parameter has an effect only if the program uses GSAM or BPAM and does not set its own value for NCP.

INPUT | OUTPUT

INPUT
specifies a BSAM data set opened for INOUT or a BDAM data set opened for UPDAT is to be processed for input only. This operand overrides the INOUT (BSAM) option or UPDAT (BDAM) option in the OPEN macro instruction to INPUT. This is useful if you only have READ access authority to the data set.

OUTPUT
specifies a BSAM data set opened for OUTIN or OUTINX is to be processed for output only. This operand overrides the OUTIN option in the OPEN macro instruction to OUTPUT or the OUTINX option in the OPEN macro instruction to EXTEND.

EXPDT(*year_day*)
specifies the data set expiration date. Specify the year and day in one of two forms:

- yyddd, where yy is the last two-digit number for the year and ddd is the three-digit number for the day of the year. The maximum value for the year is 99 (for 1999). The minimum value for the day is 000 and the maximum value is 366.

  If you specify 99365 or 99366, the system retains your data sets permanently. Do not use those dates as an expiration date. Use them as "no scratch" dates only.
- yyyy/ddd, where yyyy is the four-digit number for the year and ddd is the three-digit number for the day of the year. The slash is required. The maximum value for the year is 2155. The minimum value for the day is 000 and the maximum value is 366.

If you specify 1999/365 or 1999/366, the system retains your data sets permanently. Do not use those dates as an expiration date. Use them as "no scratch" dates only. If you code any of these special values after 1999, they will have the same effect.

If SMS is active, the expiration date might have been defined by the DATACLAS operand.

RETPD(*number_of_days*)
specifies the data set retention period in days. The value can be a five-digit decimal number with a current maximum value of 93000. If the system calculates the date as the equivalent of 1999/365 or 1999/366, the system does not use that date. Instead it uses 2000/001.

BFALN({F | D})
specifies the boundary alignment of each buffer as follows:

D
Each buffer starts on a doubleword boundary.

F
Each buffer starts on a fullword boundary that might not be a doubleword boundary.

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

OPTCD(A, B, C, E, F, H, J, Q, R, T, W, and Z or all)
specifies the following optional services that you want the system to perform. See the OPTCD subparameter of the DCB parameter in *z/OS MVS JCL Reference*.

A
specifies actual device addresses be presented in READ and WRITE macro instructions.

B
specifies the 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 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.

J
specifies the character after the carriage control character is the table reference character for that line. The table reference character tells TSO/E which character arrangement table to select when printing the line.

Q
requests the system to translate a magnetic tape from ASCII to EBCDIC or from EBCDIC to ASCII.

R
requests the use of relative block addressing.

T
requests the use of the user totaling facility.

W
requests the system to perform a validity check when data is written on a direct access device.

Z
requests the control program to shorten its normal error recovery procedure for input on magnetic tape.

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

EROPT({ACC | SKP | ABE})
specifies the option that you want to enter if an error occurs when a record is read or written. The options are:

ACC
to accept the block of records in which the error was found.

SKP
to skip the block of records in which the error was found.

ABE
to end the task abnormally.

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

S
Simple buffering

E
Exchange buffering

A
Automatic record area buffering

R
Record buffering

RECFM(A, B, D, F, M, S, T, U, and/or V)
specifies the format and characteristics of the records in the data set. The format and characteristics must be completely described by one source only. If they are not available from any source, the default is an undefined-length record. For a discussion of the formats and characteristics of the RECFM subparameter of the DCB parameter, see *z/OS MVS JCL Reference*.

Use the following values with the RECFM operand:

A
indicates the record contains ASCII printer control characters.

B
indicates the records are blocked.

D
indicates variable-length ASCII records.

F
indicates the records are of fixed-length.

M
indicates the records contain machine code control characters.

S
indicates, 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 might span more than one block. Exchange buffering, BFTEK(E), must not be used.

T
indicates the records can be written onto overflow tracks, if required. Exchange buffering, BFTEK(E), or chained scheduling, OPTCD(C), cannot be used.

U
indicates the records are of undefined-length.

V
indicates the records are of variable-length.

You can specify one or more values for this operand; at least one is required. If you use more than one value, you must separate each value with a comma or a space.

With SMS, the record format for a new data set might have been defined by the DATACLAS operand.

RECFM is mutually exclusive with RECORG.

DIAGNS(TRACE)
specifies the Open/Close/EOV trace option that gives a module-by-module trace of the Open/Close/EOV work area your 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}, {COMP | NOCOMP})
specifies the recording technique for 7 or 18 track tape as follows:

C
Data **c**onversion with odd parity (the default) and no translation (the default).

E
Even parity with no translation (the default) and no conversion (the default).

T
Odd parity (the default) and no conversion (the default). BCD to EBCDIC **t**ranslation when reading and EBCDIC to BCD translation when writing.

ET
Even parity, and no conversion (the default). BCD to EBCDIC **t**ranslation when reading and EBCDIC to BCD translation when writing.

COMP | NOCOMP
specifies whether data sets are to be compressed to save space.

This operand is mutually exclusive with KEYLEN.

KEYLEN(*bytes*)
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 operand is mutually exclusive with TRTCH.

ATTRIB command return codes

Table 7 on page 61 lists the return codes of ATTRIB command.

Table 7: ATTRIB command return codes	
Return codes	**Meaning**
0	Processing successful.
12	Processing unsuccessful. An error message has been issued.

ATTRIB command examples

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 operands that you want assigned to a data set.

- Optional services: Chained-scheduling, user totaling
- Expiration date: Dec. 31, 1985
- 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 32756 bytes
- Name of the attribute list: DCBPARMS

```
attr dcbparms optcd(c t) expdt(85365) 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 free the list so that it cannot be used again.

Known:

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

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

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

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

free attrlist(dsattrs)
```

CALL command

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

You must specify the name of the program (load module or program object) to be processed except in those situations where the CALL command assumes module "TEMPNAME". The program specified must be a member of a partitioned data set (PDS) or a partitioned data set extended (PDSE).

You can 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 1 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.

When you pass parameters to a PL/I program, precede them with a slash (/). PL/I assumes that any value before the slash is a run time option.

When you pass parameters to a C program, precede them with a slash (/) only if you have specified the EXECOPS run time option; otherwise, the slash character will be included in the characters passed as parameters.

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

CALL command in the background

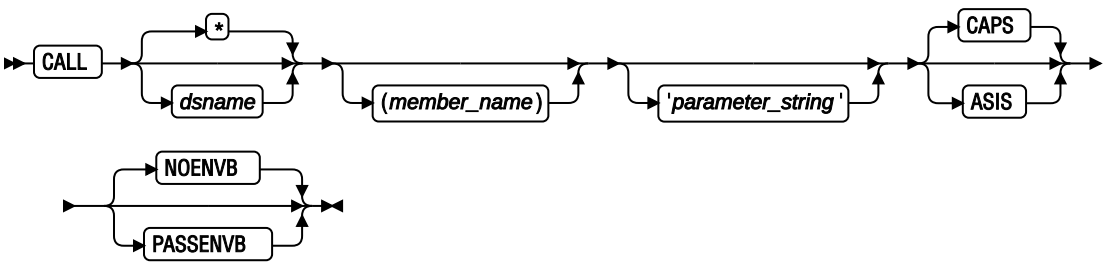
Service aids, utilities, and other programs obtaining their input from an allocated file such as SYSIN must have the input in a data set or a job stream data set (one which contains the JCL to run the job and the data itself). After the data set is created, you can use the CALL command to execute the program that accesses the SYSIN data. Figure 1 on page 62 illustrates the allocation and creation of input data sets. Information about command processing in the foreground and background is described in *z/OS TSO/E User's Guide*.

```
//EXAMP1   EXEC    PGM=IKJEFT01,DYNAMNBR=20
//SYSTSPRT DD      SYSOUT=A
//SYSTSIN  DD       *
  profile prefix(user1)
  allocate file (sysprint) dataset(*)
  allocate file(sysin) altfile(inputdd)
  call (prog1)
  allocate file(sysin) altfile(inputdd2) reuse
  call (prog2)
  free all
//INPUTDD  DD     *
  **input to prog1**
//INPUTDD2 DD     *
  **input to prog2**
/*
```

Figure 1: Allocating and creating input data sets in the background

Note: Allocating the input file to a terminal results in an I/O error message. Abnormal termination occurs when the program tries to get input from the terminal.

CALL command syntax



CALL command operands

dsname
specifies the name of a PDS or a PDSE from which the program is to be executed. If *dsname* is not fully qualified, it is assumed to be `'prefix.dsname.LOAD'`.

*****
specifies that CALL should use the standard load module search sequence for the member name.

(*member_name*)
specifies the program name to be executed. When you specify only a *member_name*, the fully—qualified dsname and *member_name*, it is assumed to be `'prefix.LOAD(member_name)'`. If *member_name*, is not specified, the member `'TEMPNAME'` is assumed.

Note: CALL command processing allocates the data set you specify and then accesses the member:

1. When allocating the data set, it is possible that the cataloged version of the data set will not be used, but rather a different copy that has already been allocated in your TSO/E session. For information about how MVS dynamic allocation may convert an existing allocation, see *z/OS MVS Programming: Authorized Assembler Services Guide*.
2. When giving control to the program, the data set you specify on the CALL command is established as a task library. The tasklib is effective for the execution of the CALL command.

parameter_string
specifies up to 100 characters of information that you want to pass to the program as a parameter string. The character string can contain DBCS characters that you delimit with shift-out (X'0E') and shift-in (X'0F') characters.

The program to be executed receives parameters according to the standard linkage conventions. These are the same conventions that will apply if the program was executed by batch job control language (JCL) and a parameter string was passed by the EXEC statement with the PARM keyword.

Some utilities accept multiple entry parameter lists; for example, to pass a list of alternate ddnames, TSO/E commands require a special multiple entry parameter list known as a command processor parameter list (CPPL). Neither of these options are supported by the CALL command, whose primary purpose is to support the execution of programs written for a batch processing environment rather than a TSO/E environment.

ASIS | CAPS

ASIS
prevents translation of a parameter list to uppercase characters. Use ASIS for programs that accept mixed case characters in a parameter list; the CALL command will not alter the parameters in any way when the ASIS option is specified.

CAPS
translates the parameter list to uppercase characters. CAPS is the default.

PASSENVB | NOENVB

PASSENVB
passes the currently active REXX environment block (ENVBLOCK) address to the invoked program in register 0. The currently active REXX ENVBLOCK is obtained from the environment to which the CALL command is directed. See “Example 6” on page 65 and “Example 7” on page 65 for uses of PASSENVB in REXX execs. This operand is:

- recognized for unauthorized programs and non-isolated environments
- ignored for authorized programs and isolated environments.

For a description of isolated environments, see *z/OS TSO/E Programming Services*. For further information about the REXX environment block see *z/OS TSO/E REXX Reference*.

NOENVB
does not pass a REXX environment block (ENVBLOCK) address. The contents of register 0 on entry to the invoked program are unpredictable. NOENVB is the default.

CALL command return codes

Table 8 on page 64 lists all the return codes of CALL command.

Table 8: CALL command return codes

Return codes	Meaning
0	Processing successful.
12	Processing unsuccessful. An error message has been issued.
Other	Return code is from the called program.

CALL command examples

Example 1

Operation: Start 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: Start a load module.

Known:

- The name of the load module: JUDAL.MYLIB.LOAD(COS1)

```
call mylib(cos1)
```

Example 3

Operation: Start a PL/I load module passing a parameter.

Known:

- The name of the load module: D58ABC.PCP.LOAD(MOD1)
- The parameter to be passed: The character string ABC

```
call 'd58abc.pcp.load(mod1)' '/abc'
```

Example 4

Operation: Start a C load module passing a parameter list in mixed case. The called program will accept the parameters as passed.

Known:

- The name of the load module: C58ABC.C.LOAD(MOD1)
- The parameter to be passed: The character string ‘a BcD’

- The NOEXECOPS option is specified so there is no need to precede the parameter list with a slash character.

```
call 'C58abc.c.load(mod1)' 'a BcD' asis
```

Example 5

Operation: Start a C load module passing a parameter list with run time options. The EXECOPS run time option must be specified.

Known:

- The name of the load module: C58ABC.C.LOAD(MOD1)
- The parameter to be passed: The character string ‘a bcd’

```
call 'C58abc.c.load(mod1)' 'NOTEST /a bcd'
```

Example 6

Operation: Start an ASM load module from a REXX exec passing the REXX environment block address to the ASM program in register 0.

Known:

- The name of the load module: STEVE.LOAD(PGM)

```
/*  REXX  */
address tso "CALL 'STEVE.LOAD(PGM)' PASSENVB"
```

Example 7

Operation: Start an ASM load module from a REXX exec invoked under IPCS.

Known:

- The name of the load module: STEVE.LOAD(PGM)
- The name of the REXX exec: STEVE.EXEC(RUNIT)

```
/*  REXX  */
address tso "ISPSTART PGM(BLSG) PARM(CMD(RUNIT))"

/*  REXX  exec RUNIT  */
address ipcs "CALL 'STEVE.LOAD(PGM)' PASSENVB"
```

For further information concerning IPCS, see *z/OS MVS IPCS User's Guide*.

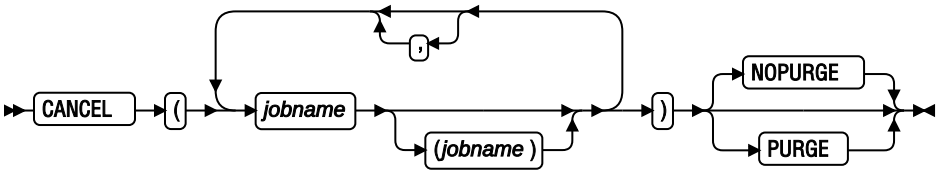
CANCEL command

Use the CANCEL command to halt processing of batch jobs that you have submitted from your terminal. A READY message is displayed at your terminal if the job has been canceled successfully. A message is also displayed at the system operator's console when a job is canceled.

CANCEL is a foreground-initiated-background (FIB) command. You must have authorization from installation management to use CANCEL. This command is generally used with the SUBMIT, STATUS, and OUTPUT commands.

Requesting an attention interrupt after issuing a CANCEL command might terminate that command's processing. In this case, you cannot resume CANCEL processing by pressing the Enter key as you can after most attention interrupts.

CANCEL command syntax



CANCEL command operands

(jobname (jobid))

Specifies the names of the jobs that you want to cancel. The job names must consist of your user identification plus 1 to 8 alphanumeric characters. However, if your installation has replaced the default IBM-supplied CANCEL exit, you might be allowed to specify different job names.

The optional job ID subfield can consist of 1 to 8 alphanumeric characters. The first character must be alphabetic or one of the special characters (#, $, or @). The job ID is a unique job identifier assigned by the job entry subsystem (JES) at the time the job was submitted to the batch system. Currently, the only valid forms of job identifiers (*jobid*) assigned by JES are:

- JOBnnnnn or Jnnnnnnn - Jobs
- STCnnnnn or Snnnnnnn - Started Tasks
- TSUnnnnn or Tnnnnnnn - TSO Users

The job ID is needed if you have submitted two jobs with the same name.

Notes:

1. When you specify a list of several job names, you must separate the job names with standard delimiters and you must enclose the entire list within parentheses.
2. Jobs controlled by the subsystems are considered started tasks and cannot be canceled by the CANCEL command.
3. If your installation uses security labels and security checking, each job has a security label associated with it. You might submit a job at a greater security label than you are currently logged on with, if you are defined to that security label. However, to cancel a job, the security label you are logged on with must be equal to or greater than the security label that the job was submitted at.
4. The CANCEL command uses the user ID as the jobname when building a JOB card, if not supplied.

NOPURGE | PURGE

NOPURGE
Specifies jobs are to be canceled if they are in execution, but output generated by the jobs remains available. If the jobs have executed, the output remains available.

PURGE
Specifies the job and its output (on the output queue) are to be purged from the system.

CANCEL command return codes

Table 9 on page 66 lists the return codes of CANCEL command.

Table 9: CANCEL Command Return Codes

Return codes	Meaning
0	Processing successful.
12	Processing unsuccessful. An error message has been issued.

CANCEL command examples

Example 1

Operation: Cancel a batch job.

Known:

- The name of the job: JE024A1

```
cancel je024a1
```

Example 2

Operation: Cancel several batch jobs.

Known: Known:

- The names of the jobs: D58BOBTA D58BOBTB(J51) D58BOBTC

```
cancel (d58bobta d58bobtb(j51) d58bobtc)
```

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.

When you specify one of your data set names to be deleted the system adds your user ID and, if possible, a descriptive qualifier. Because this can change your intended request, be careful when deleting data sets that you do not delete data sets you want to keep. For example, if you want to delete data set Z, you need to specify DELETE Z. But if data set Z did not exist and there were a data set Z.Y, data set Z.Y would not be deleted. Specify DELETE Z.Y to delete data set Z.Y. The system derives the descriptive qualifier Y from the catalog and deletes that data set.

If you want to be sure that your TSO prefix (if present) is not prepended to the data set name, place the name within single quotes. For example, if your TSO prefix is 'SAM1' and you want to delete data set Z.Y, specifying DELETE Z.Y will fail because it tries to delete 'SAM1.Z.Y'. Instead specify DELETE 'Z.Y' so that the data set named 'Z.Y' is deleted.

If more than one descriptive qualifier exists for a data set, the system will prompt you for the additional information. For example, if you have data sets Z.X and Z.Y and you issue the command DELETE Z, the system will ask you to specify qualifier X or Y.

Members of a partitioned data set and aliases for any members must each be deleted explicitly except with a PDSE member. That is, when you delete a PDS member, the system does not remove any alias names of the member. Also, when you delete a PDS or PDSE alias name, the member itself is not deleted.

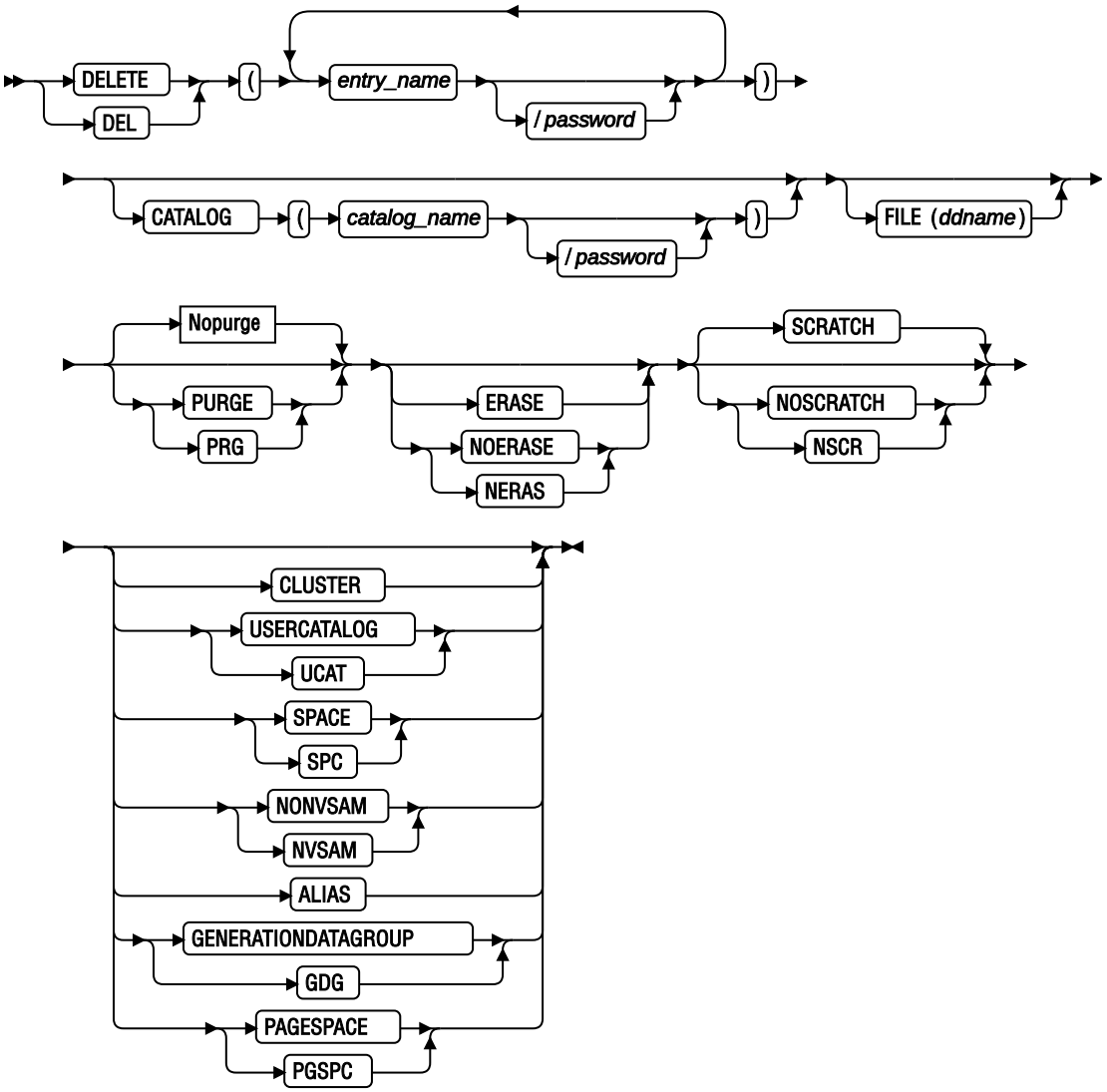
If a generation data group entry is to be deleted, any generation data sets that belong to it must have been deleted.

For MVS, the original TSO/E DELETE command has been replaced by the Access Method Services command with the same name. Note that when you delete a data set, you must also free the allocated ddnames. If you want to modify VSAM objects or use the other Access Method Services from a terminal, see *z/OS DFSMS Access Method Services Commands*. For error message information, see the MVS/ESA System Messages library.

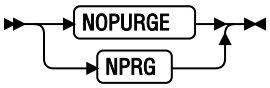
The DELETE command supports unique operand abbreviations in addition to the typical abbreviations produced by truncation. The syntax and operand explanations show these unique cases.

After you delete a protected non-VSAM data set, use the PROTECT command to delete the password from the password data set. This prevents you from having insufficient space for future entries.

DELETE command syntax



Nopurge



DELETE command operands

entry_name/password
is a required operand that names the entries in the designated catalogs to be deleted. When more than one entry is to be deleted, the list of entry_names must be enclosed in parentheses. This operand must be the first parameter following DELETE.

If you want to delete several data set entries having similar names, you can 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 are deleted. However, you can use only one asterisk per data set name. It cannot appear in the first position.

For example, assume that you have several data set entries named:

```
  1) VACOT.SOURCE.PLI.DDD
  2) VACOT.SOURCE.PLI.DDD.EEE
  3) VACOT.SOURCE.PLI

 DELETE VACOT.*   results in the deletion of no data sets.
```

```
DELETE VACOT.SOURCE.*   results in the deletion of data set #3.
DELETE VACOT.SOURCE.*.DDD   results in the deletion of data set #1.
DELETE VACOT.SOURCE.*.DDD.EEE   results in the deletion of data set #2.
```

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

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

catalog_name
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 | PRG | NOPURGE | NPRG

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

NOPURGE | NPRG
specifies the entry is not to be deleted if the retention period has not expired. When NOPURGE is coded and the retention period has not expired, the entry is not deleted. NOPURGE is the default.

ERASE | NOERASE | NERAS

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

SCRATCH | NOSCRATCH | NSCR

SCRATCH
specifies 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.

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

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

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

SPACE | SPC
specifies the entry to be deleted is a VSAM data-space entry. This operand is required if a data space is to be deleted. A data space can be deleted only if it is empty. A VSAM data space can be cataloged only in a VSAM catalog, not in an ICF catalog.

NONVSAM | NVSAM
specifies the entry to be deleted is a non-VSAM data set entry. This is an optional operand that defaults to the actual type of catalog entry. If it differs from the actual entry type, the DELETE fails.

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

GENERATIONDATAGROUP | GDG
specifies 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 | PGSPC
specifies a page space is to be deleted. A page space can be deleted only if it is inactive.

If the FILE operand is omitted, the entry_name 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.

DELETE command return codes

Table 10 on page 70 lists all the return codes of DELETE command.

Table 10: DELETE Command Return Codes

Return codes	Meaning
0	Processing successful. Informational messages might have been issued.
4	Processing successful, but a warning message has been issued.
8	Processing was completed, but specific details were bypassed.
12	Processing unsuccessful.
16	Severe error or problem encountered.

DELETE command example

Example

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

Known:

- The name of the data set to be deleted is EXAMPLE.NONVSAM.
- The prefix in the profile is D27UCAT.
- Your user ID is D27UCAT.

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

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

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

NONVSAM ensures 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

Use the EDIT command to enter data into the system. With EDIT and its subcommands, you can create, modify, store, submit, retrieve, and delete data sets with sequential or partitioned data set organization.

You cannot, however, edit an SMS-managed partitioned data set extended (PDSE). The data sets might contain:

- Source programs composed of program language statements such as PL/I, COBOL, FORTRAN, and so on.
- Data used as input to a program.
- Text used for information storage and retrieval.
- Commands, subcommands, CLIST statements and data or all.
- Job control language (JCL) statements for background jobs.

The EDIT command supports 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 originally 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 are warned that the data set will be saved without control characters, and you can choose 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, you cannot enter SAVE.

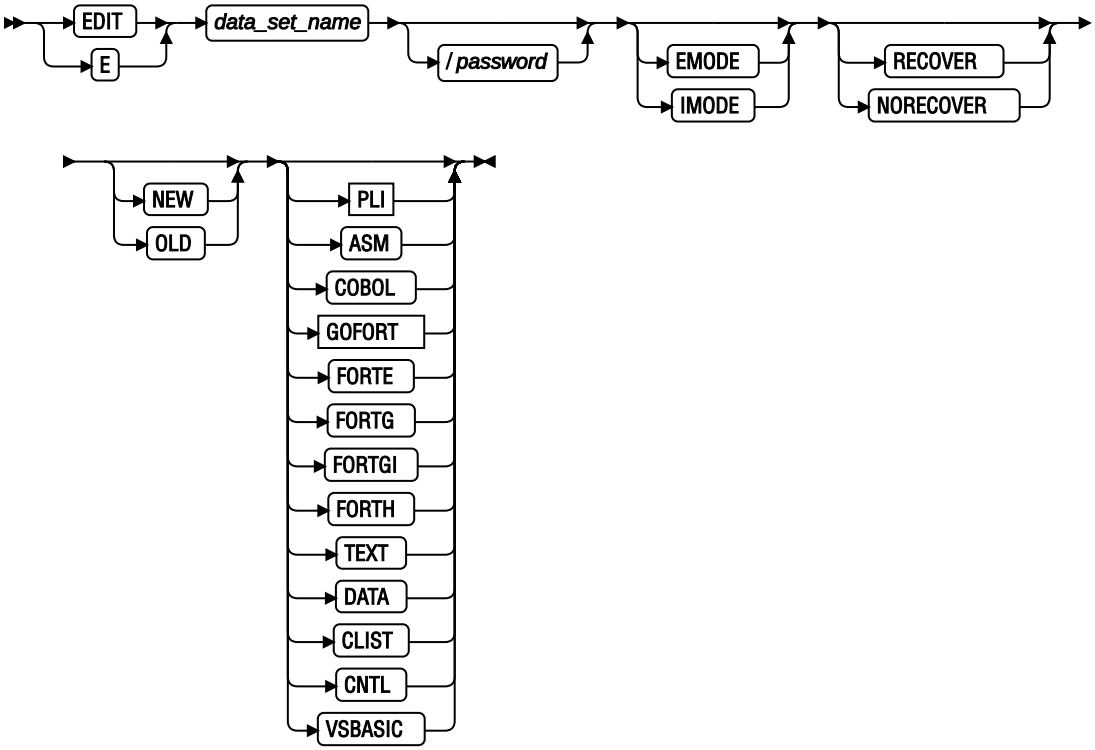
When you have finished editing a data set with a variable-blocked record format, each record (line) is padded with blanks to the end of the record. When you save the data set, the blanks are eliminated and the length adjusted accordingly.

EDIT does not serialize on a member of a PDS, thus, it is possible for multiple users to edit the same member of the same PDS at the same time. This can result in only one user's changes being saved in the data set.

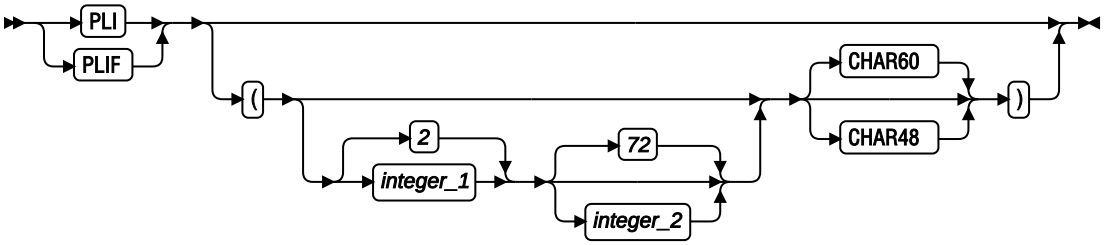
When using REXX to invoke PROMPT, PUTGET or subcommand INPUT processing in the background, you must use the REXX PROMPT function to activate input from the REXX stack. Regardless of the REXX PROMPT function setting, the REXX stack is queried first in foreground. If the system does not find any input, the TERMINAL is queried to satisfy the input. Regardless of the PROMPT setting, EDIT reads (“prompts”) the terminal for input. Only the REXX stack is queried for the command response input in background. In automated background processing, you can use the REXX PROMPT function setting to control whether the REXX stack is available for command or subcommand input or not.

For more information about using line mode edit, see Table 6. How EDIT Subcommands Affect the Line Pointer Value in Appendix B, Using Line Mode Edit of *z/OS TSO/E User's Guide*.

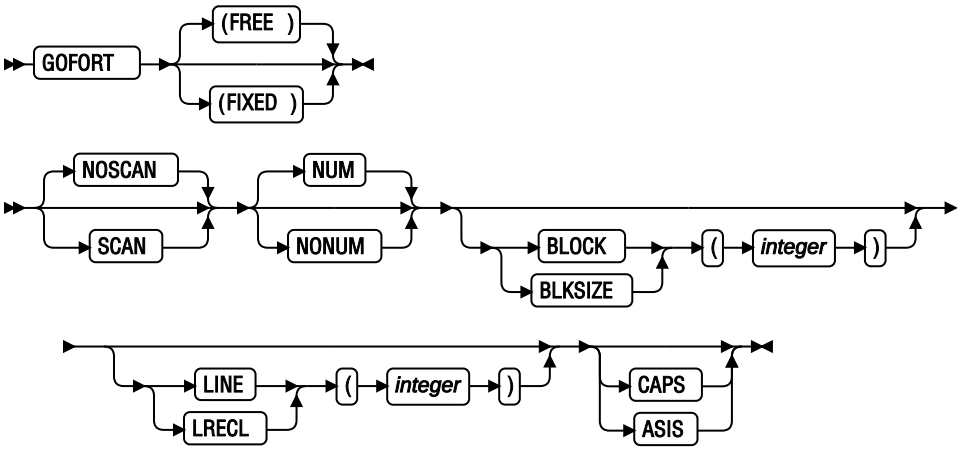
EDIT command syntax



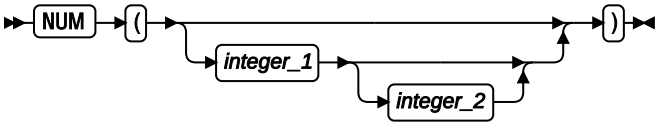
PLI



GOFORT



NUM



EDIT command operands

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

If you enter the name of a sequential data set, but the data set is actually found to be a partitioned data set, the member name TEMPNAME is assumed. See also the description of the OLD operand below.

password
specifies the password associated with the *data_set_name*. If the password is omitted and the data set is password protected, you are prompted for the data set's password. Read protected partitioned data sets prompt for the password twice, provided it is not entered on the EDIT command, or is not the same password as your LOGON user ID password.

EMODE | IMODE

EMODE
specifies the initial mode of entry is edit mode. This is the default for OLD data sets. See *z/OS TSO/E User's Guide*, for more information about using edit mode.

IMODE
specifies the initial mode of entry is input mode. This is the default for NEW or empty data sets. See *z/OS TSO/E User's Guide*, for more information about using input mode.

RECOVER | NORECOVER

RECOVER
specifies that you intend to recover an EDIT work file containing the data set named on the EDIT command as the data set to be edited. You are placed in edit mode. This operand is valid only when your profile has the RECOVER attribute. See *z/OS TSO/E User's Guide*, for more information.

NORECOVER
specifies that you do not want to recover a work file, even if a recoverable work file exists. Any existing work files will be reused to hold data set information for the current edit session. As a result, any recovery information from a previous session will be lost.

NEW | OLD

NEW
specifies the data set named by the first operand does not exist. If an existing cataloged data set already has the data set name that you specified, the system notifies you when you try to save it. Otherwise, the system allocates your data set when you save it. If you specify NEW without specifying a member name, a sequential data set is allocated for you when you save it. If you specify NEW and include a member name, the system allocates a partitioned data set and creates the indicated member when you try to save it.

OLD
specifies 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 are notified and you have to reenter the EDIT command. If you specify OLD without specifying a member name, the system assumes that your data set is sequential. If the data set is, in fact, a partitioned data set, the system assumes that the member name is TEMPNAME. If you specify OLD and include a member name, the system notifies you if your data set is not partitioned.

Note: Specifying OLD will not prevent other users from using EDIT to update the same member in the same partitioned data set (PDS) at the same time. OLD informs the EDIT command that the data set already exists - it does not provide protection equivalent to specifying DISP=OLD for the data set.

If you do not specify OLD or NEW, the system uses a tentative default of OLD. If the data set name or member name that you specified cannot be located, the system defaults to NEW.

Any user-defined data set type is also a valid data set type operand and can have subfield parameters defined by your installation (see Table 11 on page 76, note 4).

PLI
specifies 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 can be for the PL/I Optimizing compiler or the PL/I Checkout compiler.

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

integer_1* and *integer_2
specify the column boundaries for your input statements. These values are applicable only when you request syntax checking of a data set for which the PLIF operand has been specified. The position of the first character of a line, as determined by the left margin adjustment on your terminal, is column 1. The value for *integer_1* specifies the column where each input statement is to begin. The statement can extend from the column specified by *integer_1* up to and including the column specified as a value for *integer_2*. If you omit *integer_1*, you must omit *integer_2*. The default values are columns 2 and 72. However, you can omit *integer_2* without omitting *integer_1*.

CHAR48 | CHAR60

CHAR48
specifies the PL/I source statements are written using the character set that consists of 48 characters.

CHAR60
specifies 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 the data set identified by the first operand is for assembler language statements.

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

CLIST
specifies the data set identified by the first operand is for a CLIST and contains TSO/E commands, subcommands, and CLIST statements as statements or records in the data set. The data set is assigned line numbers.

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

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

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

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

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

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

GOFORT(FREE | FIXED)
specifies the data set identified by the first operand is for statements that are suitable for processing by the Code and Go FORTRAN licensed program. If you enter the descriptive qualifier without a data

set type, the data set type default is GOFORT(FREE). If you do not specify a FORTRAN language level, GOFORT is the default value. FREE specifies the statements are of variable-lengths and do not conform to set column requirements. If you do not specify FREE or FIXED, FREE is the default. FIXED specifies statements adhere to standard FORTRAN column requirements and are 80 bytes long.

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

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

- The *data_set_name* operand does not follow data set naming conventions (that is, it is enclosed in quotation marks).
- The *data_set_name* operand is a member name only (that is, it is enclosed in parentheses).
- The *data_set_name* operand does not include a descriptive qualifier or the descriptive qualifier is such that EDIT cannot determine the data set type.

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

Note: If PLI is the descriptive qualifier, the data set type default is PLI. To use data set types GOFORT, FORTGI, or FORTH, you must enter the data set type operand to save it.

SCAN | NOSCAN

SCAN
specifies 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 FORTGI or FORTH.

If your installation specified a syntax checker after system generation, user-defined data set types can also use the SCAN operand.

NOSCAN
specifies syntax checking is not to be performed. NOSCAN is the default.

NUM(*integer_1 integer_2*) | NONUM

NUM(*integer_1 integer_2*)
specifies lines of the data set records are numbered. You can specify *integer_1* and *integer_2* for ASM type data sets only. *integer_1* specifies, in decimal, the starting column (73-80) of the line number. *integer_2* specifies, in decimal, the length (8 or less) of the line number. *integer_1* plus *integer_2* cannot exceed 81. If *integer_1* and *integer_2* are not specified, the line numbers assume appropriate default values.

NUM is the default.

NONUM
specifies your data set records do not contain line numbers. Do not specify this operand for the VSBASIC and CLIST data set types because they must always have line numbers.

BLOCK(*integer*) | 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 defaults according to the type of data set being created. The IBM-supplied default values for the block sizes are described in Table 11 on page 76. To modify those default values, see *z/OS TSO/E Customization*. 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.

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

LINE(*integer*) | LRECL(*integer*)

specifies the length of the records to be created for a new data set. Specify this operand only when creating a new data set or editing an empty old data set. The new data set is 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, COBOL, or CNTL, the integer must be 80. If this operand is omitted, the length defaults according to the type of data set being created. The IBM-supplied default values are described in Table 11 on page 76. To modify those default values, see *z/OS TSO/E Customization*. Use this operand with the BLOCK or BLKSIZE operand.

CAPS | ASIS

CAPS

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

ASIS

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

Table 11: EDIT command: default values for LINE or LRECL and BLOCK or BLKSIZE operands.

Data set type	DSORG	LRECL		Block size		Line numbers		
		LINE(n)		BLOCK(n)		NUM(n,m)	CAPS/ASIS	
		Default	Specif.	Default	Specif. (Note 1)	Default(n,m) Specif.	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
FORTE	PS/PO	80	<=255	3120	<=default	Last 8	CAPS	Yes
FORTG	PS/PO	80	<=255	3120	<=default	Last 8	CAPS	Yes
FORTGI	PS/PO	255	=80	400	<=default	Last 8	CAPS	Yes
FORTH	PS/PO	255	=80	400	<=default	Last 8	CAPS	Yes
GOFORT	PS/PO	255		3120	<=default	First 8	CAPS	Yes
(Or user supplied data set type - see Note 4)								
PLI	PS/PO	104	<=100	400	<=default	(Note 3)	CAPS	No
PLIF	PS/PO	80	<=100	400	<=default	Last 8	CAPS	Yes
TEXT	PS/PO	255	(Note 2)	3120	<=default	(Note 3)	ASIS	No
VSBASIC	PS/PO	255	=80	3120	<=32,760	First 5	CAPS	Yes

Table 11: EDIT command: default values for LINE or LRECL and BLOCK or BLKSIZE operands. (continued)

Data set type	DSORG	LRECL		Block size		Line numbers		
		LINE(n)		BLOCK(n)		NUM(n,m)	CAPS/ASIS	
		Default	Specif.	Default	Specif. (Note 1)	Default(n,m) Specif.	Default	CAPS Required

Notes

1. IBM supplies the default values. For information about how to modify the default values, see *z/OS TSO/E Customization*.
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 operand is omitted, variable-length records are created.
3. The line numbers are 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. You can modify the user-defined data set types along with any of the data sets shown above *after* 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 about how to modify the EDIT macro, see *z/OS TSO/E Customization*.

   When you edit a data set type you defined yourself, the system uses the data set type as the descriptor (right-most) qualifier. You cannot override any data set types that have been defined by IBM. The EDIT command processor supports data sets that have the following attributes:

   Data Set Organization:
   Must be either sequential or partitioned

   Record Format:
   Fixed or variable

   Logical Record Size:
   Less than or equal to 255 characters

   Block Size:
   User specified-must be less than or equal to track length

   Sequence Number:
   V type--First 8 characters

   F type--Last 8 characters

EDIT command return codes

Table 12 on page 77 lists the return codes of EDIT command.

Table 12: EDIT command return codes

Return codes	Meaning
0	Processing successful.
12	Processing unsuccessful.

EDIT command examples

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 (where WRR05 is the user ID) 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 (where WRR05 is the user ID) 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 uppercase.

```
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 uppercase.
- Syntax checking is not applicable.

```
e manhrs.data
```

Example 4

Operation: Create a data set for a CLIST.

Known:

- The user supplied name for the data set: CMDPROC

```
e cmdproc new clist
```

EDIT subcommands (overview)

Use the subcommands while in edit mode to edit and modify 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 like that for a command. Table 13 on page 79 contains a summary of each subcommand's function.

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

Note: Invocation of subcommands FORMAT, MERGE, RUN, and SUBMIT, without specifying data set name(s), causes the EDIT command to allocate a new and cataloged data set with the name of 'prefix.subcommand.date.time'. The data set is deleted when the subcommand completes. If you are running with profile NOPREFIX, you might want to set PREFIX to the user ID.

[2] Available as an optional licensed program.

Table 13: Subcommands and functions of the EDIT command

Subcommand	Function
ALLOCATE	Allocates data sets and file names.
ATTRIB	Builds a list of attributes for non-VSAM data sets.
BOTTOM	Moves the pointer to the last record in the data set.
CHANGE	Alters the contents of a data set.
CKPOINT	Protects input or modifications to 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 CLIST or REXX exec.
FIND	Locates a character string.
FORMAT Available as an optional licensed program.	Formats and lists data.
FREE	Releases previously allocated data sets.
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 [2]	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.

EDIT—ALLOCATE subcommand

Use the ALLOCATE subcommand to dynamically allocate the data sets required by a program that you intend to execute. For a description of the ALLOCATE command syntax and function, see the "ALLOCATE command" on page 8.

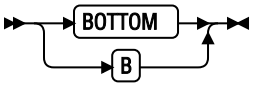
EDIT—ATTRIB subcommand

The ATTRIB subcommand of EDIT performs the same function as the ATTRIB command without leaving edit mode. For a description of the ATTRIB command syntax and function, see the "ATTRIB command" on page 53.

EDIT—BOTTOM subcommand

Use the BOTTOM subcommand to change the current line pointer to the last line of the data set you are editing or to contain a zero value (if the data set is empty). This subcommand can be useful when following subcommands such as INPUT or MERGE must be at the end of the data set.

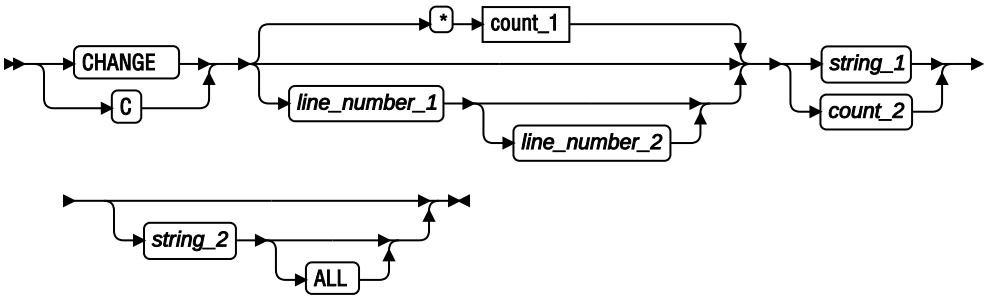
EDIT—BOTTOM subcommand syntax



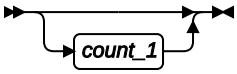
EDIT—CHANGE subcommand

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

EDIT—CHANGE subcommand syntax



count_1



EDIT—CHANGE subcommand operands

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 *string_1*.

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

count_1
specifies the number of lines that you want to change, starting at the position indicated by the asterisk (*).

string_1
specifies a sequence of characters that you want to change. The sequence must be (1) enclosed within single quotation marks, or (2) preceded by an extra character which serves as a special delimiter. The extra character may be any printable character other than a single quotation mark (apostrophe), number, blank, tab, comma, semicolon, parenthesis, or asterisk. The hyphen (-) and plus (+) signs can be used, but should be avoided because of possible confusion with their use in continuation. If the first character in the character string is an asterisk (*), do not use a slash (/) as the extra character. (TSO/E interprets the /* as the beginning of a comment.) The extra character must not appear in the character string. Do not put a standard delimiter between the extra character and the string of characters unless you intend the delimiter to be treated as a character in the character string.

If *string_1* is specified and *string_2* is not, the specified characters are displayed at your terminal up to (but not including) the sequence of characters that you specified for *string_1*. You can then complete the line.

Note: If you are changing a string to a string of larger size, EDIT inserts the larger string and attempts to preserve the rest of the line, including spaces.

string_2
specifies a sequence of characters that you want to use as a replacement for *string_1*. Like *string_1*, *string_2* must be (1) enclosed within single quotation marks, or (2) preceded by a special delimiter. This delimiter must be the same as the extra character used for *string_1*. Optionally, this delimiter can also immediately follow *string_2*.

Note: If you are changing a string to a string of larger size, EDIT inserts the larger string and attempts to preserve the rest of the line, including spaces.

ALL
specifies every occurrence of *string_1* within the specified line or range of lines are replaced by *string_2*. If this operand is omitted, only the first occurrence of *string_1* is replaced with *string_2*.

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

If the special delimiter form is used, *string_2* must be followed by the delimiter before typing the ALL operand.

count_2
specifies a number of characters to be displayed at your terminal, starting at the beginning of each specified line.

Quoted-String notation

As indicated previously, instead of using special delimiters to indicate a character string, you can use paired single quotation marks (apostrophes) to accomplish the same function with the CHANGE subcommand. The use of single quotation marks as delimiters for a character string is called *quoted-string notation*. Following are the rules for quoted-string notation for the *string_1* and *string_2* operands:

- Do not use both special-delimiter and quoted-string notation in the same subcommand.
- Enclose each string with single quotation marks; for example, `'This is string 1' 'This is string 2'`. Quoted strings must be separated with a blank.
- Use two single quotation marks to represent a single quotation mark within a character string; for example, `'pilgrim's progress'`.
- Use two single quotation marks to represent a null string; for example,".

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

Function	***Special-delimiter notation**	**Quoted-string notation**
Replace	!ab!cde!	‘ab’‘cde’

Function	*Special-delimiter notation	Quoted-string notation
Delete	!ab!!or!ab!	‘ab’ ''
Print up to	!ab	‘ab’
Place in front of	!!cde!	'' ‘cde’

Note: * - using the exclamation point (!) as the delimiter.

Note the following:

1. Choose the form that best suits you (either special-delimiter or quoted-string) and use it consistently. It will help you use the subcommand.
2. If you cause an attention interruption during the CHANGE subcommand, your data set might not be completely changed. You should list the affected parts 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 *count_2* operand.
- When you enter two numbers and no other operands, the system assumes that they are *line_number_1* and *count_2*.
- 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 *string_1*.
- When you enter three operands and they are all numbers, the system assumes that they are *line_number_1*, *line_number_2*, and *count_2*.
- When you enter three operands and the first two are numbers, but the last begins with a character that is not a number, the system assumes that they are *line_number_1*, *line_number_2*, and *string_1*.

EDIT—CHANGE subcommand examples

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 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 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 *string_1* and *string_2* examples (`i.e.` and `e.g.`) are treated as characters.

Example 3

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

Known:

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

```
change 143 18
```

This form of the subcommand causes the first 18 characters of line number 143 to be displayed at your terminal. You complete the line by typing the new information and enter the line by pressing the Enter key. All of your changes are 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 is not 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 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 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 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' '
```

Example 8

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

Known:

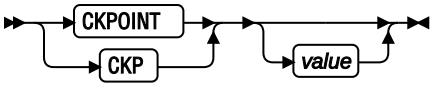
- The sequence of characters to be deleted: weekly

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

EDIT—CKPOINT subcommand

Use the CKPOINT subcommand to protect input or modifications to a data set during an EDIT session. All changes are placed in a work file (utility data set) created by EDIT and are accessible to you if an abnormal termination occurs. The purpose of this subcommand is to eliminate the need for specifying the SAVE subcommand of EDIT to preserve changes.

EDIT—CKPOINT subcommand syntax



EDIT—CKPOINT subcommand operand

value

specifies the intervals (number of line modifications or input lines) at which a checkpoint is taken. You can use the value operand in one of three ways:

- By specifying a decimal value from 1 to 9999 to be used as the checkpoint intervals.
- By specifying a decimal value of zero to terminate interval checkpointing.
- By not specifying a value, causing a checkpoint to be taken. This can be done even though you have already requested interval checkpointing. Checkpointing does not stop in this case, but continues after reaching the previously set interval value.

A line is considered modified if it is inserted, deleted, or changed. Issuing the CHANGE subcommand repeatedly and specifying the same line is equivalent to modifying the line once the CHANGE subcommand is executed.

EDIT—CKPOINT subcommand examples

Example 1

When the CKPOINT subcommand is issued without operands, EDIT ensures that all changes or modifications made up to this point are reflected in the work file. To do this, enter:

```
CKPOINT
```

or

```
CKP
```

Example 2

When the CKPOINT subcommand is issued with an operand value of 1 to 9999, a checkpoint is taken immediately and at requested intervals specified by the operand value until termination. To do this, enter:

```
CKPOINT value
```

or

```
CKP value
```

Example 3

When interval checkpointing is in effect and you want to alter the active value, reissue the CKPOINT subcommand inserting the new value like this:

```
CKPOINT new_value
```

or

```
CKP new_value
```

Example 4

To terminate interval checkpoint, issue the CKPOINT subcommand with a zero value. The entry is:

```
CKPOINT 0
```

or

```
CKP 0
```

EDIT—COPY subcommand

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

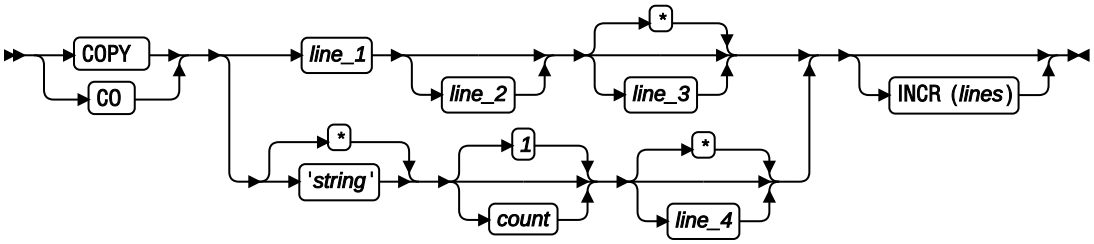
The target line cannot be within the source area, with the exception that the target line (the first line of the target area) can overlap the last line of the source area.

On 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.

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.

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.

EDIT—COPY subcommand syntax



EDIT—COPY subcommand operands

line_1
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.

line_2
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 *line_2*. If *line_2* is not entered, the value defaults to the value of *line_1*; that is, the source becomes one line. Do not enter an asterisk for *line_2*.

If COPY is followed by two line number operands, the system assumes them to represent *line_1* and *line_3*, and defaults *line_2* to the value of *line_1*.

line_3
specifies the target line number; that is, the line at which the copied-to data area starts. If the *line_3* value corresponds to an existing line, the target line is changed to *line_3* + INCR(*lines*) and either becomes a new line or displaces an existing line at that location. When 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 *line_3* 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 *line_3* set to *. Note that *line_3* defaults to *.

The value of *line_3* should not fall in the range from *line_1* to *line_2*. The target line must not be in the range being copied. Exception: *Line_3* can be equal to *line_2*.

*****
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.

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

'string'
specifies a sequence of alphanumeric characters with a maximum length equal to or less than the logical record length of the data set you are editing. 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. Enter 1-8 decimal digits, but not zero (0) or asterisk (*). The default for *count* depends on what is specified for '*string*' ('*string*' or *). If '*string*' is specified and *count* is left blank, the default for *count* is one (1). For example, if you specify:

```
COPY 'xyz' 99
```

the *count* default is one (1).

However, if you specify an asterisk (*) for the 'string', the next operand is treated as the count operand. For example, if you specify:

```
COPY * 99
```

the count is 99.

line_4
applies to both numbered and unnumbered data sets. For unnumbered data sets, *line_4* specifies the target line (the line at which the copied-to data area starts) as a relative line number (the *n*th line in the data set). For numbered data sets, *line_4* is specified the same as *line_3*. Specifying zero (0) for *line_4* 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 *line_4* set to *. The default for *line_4* is *. However, if '*string*' is specified and count is left blank, the operand following '*string*' is treated as the count operand and the *line_4* default (*) is used.

For example, if you specify :

```
COPY 'xyz' 99
```

the count is 99 and *line_4* is *.

EDIT—COPY subcommand examples

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

Example 1

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

Known:

- Data set contains lines 10 through 120.
- Current line pointer is at 50.
- EDIT provides an increment of 10.

```
Before:              Enter:                  After:

0010    A            copy     50  50  50     0010    A
0020    BB                                   0020    BB
0030    CCC          or                      0030    CCC
0040    DDDD                                 0040    DDDD
0050    EEEEE        copy     50  50         0050    EEEEE
0060    FFFFFF                         CLP   0060    EEEEE
0070    GGGGGGG      or                      0061    FFFFFF
0080    HHHHHHHH                             0070    GGGGGGG
0090    IIIIIIIII    copy     50             0080    HHHHHHHH
0100    JJJJJJJJJJ                           0090    IIIIIIIII
0110    KKKKKKKKKK   or                      0100    JJJJJJJJJJ
0120    LLLLLLLLLLL                          0110    KKKKKKKKKK
                     copy                    0120    LLLLLLLLLLL
```

```
                    or

                    copy     'ee'
```

Example 2

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

Known:

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

```
Before:             Enter:                  After:

A                   copy     *  1  *        A
BB                                          BB
CCC                 or                      CCC
DDDD                                        DDDD
EEEEE               copy     *  1           EEEEE
FFFFFF                                      FFFFFF
GGGGGGG             or                      GGGGGGG
HHHHHHHH                          CLP       GGGGGGG
IIIIIIIII           copy     *              HHHHHHHH
JJJJJJJJJJ                                  IIIIIIIII
KKKKKKKKKKK         or                      JJJJJJJJJJ
LLLLLLLLLLL                                 KKKKKKKKKKK
                    copy                    LLLLLLLLLLL

                    or

                    copy     'gg'
```

Example 3

Operation: 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 an 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    LLLLLLLLLLL                           0110    KKKKKKKKKKK
```

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'  1 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.

```
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  KKKKKKKKKK                                 0105    A
0120  LLLLLLLLLLL                                0110    BB
                                        CLP      0115    CCC
                                                 0116    KKKKKKKKKK
                                                 0120    LLLLLLLLLLL
```

Example 6

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

Known:

- Data set contains lines 10 through 120.
- Source lines are 20 through 50.
- Target area starts at line 50.
- EDIT provides an 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    KKKKKKKKKK                          0092    GGGGGGG
0120    LLLLLLLLLLL                         0093    HHHHHHHH
                                            0094    IIIIIIIII
                                            0100    JJJJJJJJJJ
                                            0110    KKKKKKKKKK
                                            0120    LLLLLLLLLLL
```

Example 7

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

Known:

- Data set contains lines 10-120.
- Source range is line 70-110.
- Target location is line 50.
- Increment is 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    GGGGGG
0070  GGGGGGG                                     0070    HHHHHHH
0080  HHHHHHHH                                    0080    IIIIIIII
0090  IIIIIIIII                                   0090    JJJJJJJJJ
0100  JJJJJJJJJ                            CLP    0100    KKKKKKKKK
0110  KKKKKKKKKK                                  0101    FFFFFF
0120  LLLLLLLLLLL                                 0102    GGGGGGG
                                                  0103    HHHHHHHH
                                                  0104    IIIIIIIII
                                                  0105    JJJJJJJJJ
                                                  0110    KKKKKKKKKK
                                                  0120    LLLLLLLLLLL
```

Example 8

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

Known:

- Data set contains lines 10 through 120.
- Line 0 does not exist.
- Source lines are 80, 90, and 100.
- Target area starts at line 0.

```
Before:              Enter:                        After:

0010  A            top                           0000    HHHHHHHH
0020  BB           copy  80  100 *  incr(50)     0050    IIIIIIII
0030  CCC                                  CLP   0100    JJJJJJJJJ
0040  DDDD         or                            0101    A
0050  EEEEE                                      0102    BB
0060  FFFFFF       copy  80  100 0  incr(50)     0103    CCC
0070  GGGGGGG                                    0104    DDDD
0080  HHHHHHHH                                   0105    EEEEE
0090  IIIIIIIII                                  0106    FFFFFF
0100  JJJJJJJJJ                                  0107    GGGGGGG
0110  KKKKKKKKKK                                 0108    HHHHHHHH
0120  LLLLLLLLLLL                                0109    IIIIIIIII
                                                 0110    JJJJJJJJJ
                                                 0111    KKKKKKKKKK
                                                 0120    LLLLLLLLLLL
```

Example 9

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

Known:

- Data set contains lines 0 through 120.
- Line 0 contains data.
- Source lines are 80, 90, and 100.

- Target area starts at line 0.

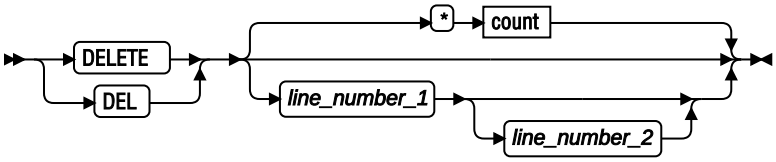
```
Before:              Enter:                        After:

0000  ZIP          top                           0050    HHHHHHHH
0010  A            copy   80  100 *  incr(50)    0100    IIIIIIIII
0020  BB                                  CLP    0150    JJJJJJJJJJ
0030  CCC          The attempt to copy 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    HHHHHHHH
0120  LLLLLLLLLLL                                0160    IIIIIIIII
                                                 0161    JJJJJJJJJJ
                                                 0162    KKKKKKKKKK
                                                 0163    LLLLLLLLLLL
                   Note:  An entry of
                   copy   80  100 0 incr(50)
                   produces the results
                   shown at right.  The target   0000    ZIP
                   data is inserted between      0050    HHHHHHHH
                   line 0 and the remainder      0100    IIIIIIIII
                   of the data set.         CLP  0150    JJJJJJJJJJ
                                                 0151    A
                                                 0152    BB
                                                 0153    CCC
                                                 0154    DDDD
                                                 0155    EEEEE
                                                 0156    FFFFFF
                                                 0157    GGGGGGG
                                                 0158    HHHHHHHH
                                                 0159    IIIIIIIII
                                                 0160    JJJJJJJJJJ
                                                 0161    KKKKKKKKKK
                                                 0162    LLLLLLLLLLL
```

EDIT—DELETE subcommand

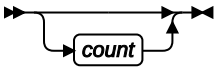
Use the DELETE subcommand to remove one or more records from the data set you are editing.

Upon completion of the delete operation, the current line pointer points to the line that preceded the deleted line. If the first line of the data has been deleted, the current line pointer is set to zero.

EDIT—DELETE subcommand syntax



count



EDIT—DELETE subcommand operands

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 the first line to be deleted is the line indicated by the current line pointer in the system. If no line is specified, then this is the default.

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

EDIT—DELETE subcommand examples

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 preceding command combinations or abbreviations 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 zeros 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:

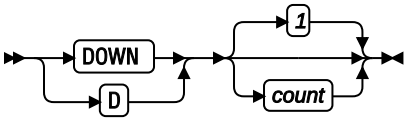
- The data set contains less than 100 lines and is not line-numbered.

```
top
delete * 100
```

EDIT—DOWN subcommand

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.

EDIT—DOWN subcommand syntax



EDIT—DOWN subcommand operand

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.

EDIT—DOWN subcommand examples

Example 1

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

```
down
```

or

```
d
```

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
```

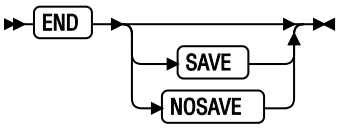
or

```
d 18
```

EDIT—END subcommand

Use the END subcommand to terminate the EDIT command. You can use this subcommand with or without the optional operands SAVE or NOSAVE. In either case, the EDIT command terminates processing. If you have modified your data set and have not entered the SAVE subcommand or the SAVE/NOSAVE operand on END, the system asks you if you want to save the data set. If you want to save the data set, reply SAVE. If you do not want to save the data set, reply END.

EDIT—END subcommand syntax



There are no defaults. If you do not specify an operand or SAVE after the last modification, you are prompted by the system.

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

EDIT—END subcommand operands

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

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

EDIT—EXEC subcommand

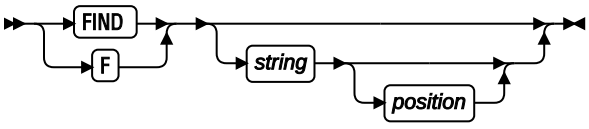
Use the EXEC subcommand to execute a CLIST or REXX exec. For a description of the EXEC command syntax and function, see the "EXEC command" on page 120.

Specify only REXX instructions in the REXX exec. Specify only EDIT subcommands and CLIST statements in the CLIST. You cannot specify TSO/E commands in the CLIST or REXX exec *until* you specify END to terminate EDIT.

EDIT—FIND 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.

EDIT—FIND subcommand syntax



EDIT—FIND subcommand operands

If you do not specify any operands, the operands you specified last with FIND are used. The search for the specified string begins at the line following the current line. If you issue the TOP subcommand, the search for the specified string begins with the second line of the data set. Successive use of the FIND subcommand without operands allows you to search a data set, line by line.

string
specifies the sequence of characters (the character string) that you want to locate. You must precede this sequence of characters with an extra character that serves as a special delimiter. The extra character can be any printable character *other* than a number, apostrophe, semicolon, blank, tab, comma, parenthesis, or asterisk. Do not use the extra character in the character string or 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 quotation marks (apostrophes) to accomplish the same function with the FIND subcommand. The use of single quotation marks as delimiters for a character string is called quoted-string notation. Following are the rules for quoted-string notation for the string operand:

1. Enclose a string within single quotation marks; for example, ‘string character’.
2. Use two single quotation marks to represent a single quotation mark within a character string; for example, ‘pilgrims's progress’.
3. Use two single quotation marks to represent a null string; for example, ''.

position
specifies the column within each line at which you want the comparison for the string to be made. This operand specifies the starting column of the field to which the string is compared in each line. If you want to consider a string starting in column 6, you must specify the digit 6 for the position operand. For COBOL data sets, the starting column is calculated from the end of the six-digit line number. If you want to consider a string starting in column 8, you must specify the digit 2 for this operand. When you use this operand with the special-delimiter form of notation for string, you must separate it from the string operand with the same special delimiter as the one preceding the string operand.

EDIT—FIND subcommand examples

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
```

EDIT—FREE subcommand

Use the FREE subcommand of EDIT to release (deallocate) previously allocated data sets that you no longer need. For a description of the FREE command syntax and function, see the “FREE command” on page 138.

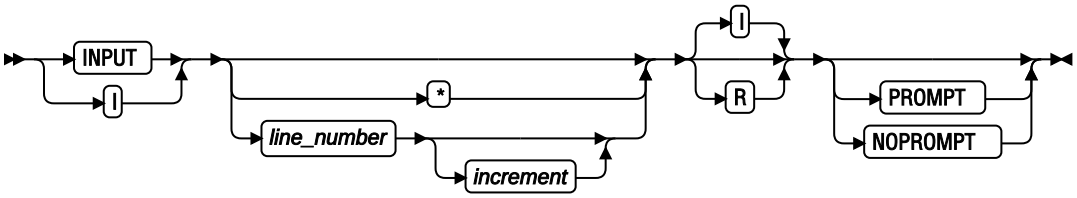
EDIT—HELP subcommand

Use the HELP subcommand to obtain the syntax and function of EDIT subcommands. For a description of the HELP command syntax and function, see the “HELP command” on page 143.

EDIT—INPUT subcommand

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

EDIT—INPUT subcommand syntax



EDIT—INPUT subcommand operands

line_number
specifies the line number and location for the first new line of input. If no operands are specified, input data is 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 is used.

*****
specifies the next new line of input either replaces or follows 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. If you fail to specify either a line number or an asterisk, this operand is ignored. If the specified line already exists, the old line is replaced by the new line. If the specified line is vacant, the new line is inserted at that location. If the increment is greater than 1, all lines between the replacement lines are deleted.

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

PROMPT | NOPROMPT

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 setting is:

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

NOPROMPT
specifies that you do not want to be prompted.

EDIT—INPUT subcommand examples

Example 1

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

Known:

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

```
input 2 2 r prompt
```

The display at your terminal will resemble the following with your input in lowercase and the system's response in uppercase.

```
edit quer cobol old
EDIT
input 2 2 r prompt
INPUT
00002 identification division
```

```
00004 program-id.query
00006
```

Example 2

Operation: Insert data in an existing data set.

Known:

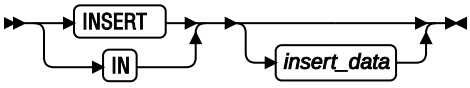
- The data set contains text for a report.
- The data set does not have line numbers.
- The ability to replace lines is not necessary.
- The first input data is "New research and development activities will", which is to be placed at the end of the data set.
- The display at your terminal will resemble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

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

EDIT—INSERT subcommand

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 current line pointer in the system. If no operands are specified, input data is placed in the data set line following the current line. You can change the position pointed to by the line pointer by using the BOTTOM, DOWN, TOP, UP, and FIND subcommands.

EDIT—INSERT subcommand syntax



EDIT—INSERT subcommand operand

insert_data
specifies the complete sequence of characters that you want to insert into the data set at the location indicated by the current 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.

EDIT—INSERT subcommand examples

Example 1

Operation: Insert a single line into a data set.

Known:

- The line to be inserted is:

```
UCBFLG 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.
- You are 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 is inserted:

```
up 3
```

The INSERT subcommand is now entered:

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

The display at your terminal shows the following:

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

Example 2

Operation: Insert several lines into a data set.

Known:

- The data set contains line numbers.
- The inserted lines are to follow line number 00280.
- The current line pointer points to line number 00040.
- You are 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 is inserted:

```
down 24
```

The INSERT subcommand is now entered:

```
insert
```

The system responds with:

```
INPUT
```

The lines to be inserted are now entered.

The display at your terminal shows the following:

```
down 24
insert
INPUT
00281 j.w.house 13-244831 24.73
00282 t.n.howard 24-782095 3.05
00283 b.h.ireland 40-007830 104.56
```

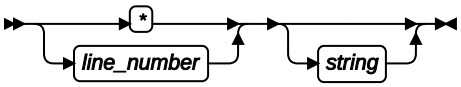
New line numbers are generated in sequence beginning with the number one greater than the one pointed to by the current line pointer. When no line can be inserted, you are notified. No re-sequencing is done by the system.

EDIT—insert/replace/delete function

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, indicate the location and the new data. To delete a line, 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.

EDIT—insert/replace/delete function syntax



EDIT—insert/replace/delete function operands

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

*****
specifies 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 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 replaces the existing line with the specified sequence of characters or, if there is no existing data at the location, the system inserts the sequence of characters into the data set at the specified location.

EDIT—insert/replace/delete function examples

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

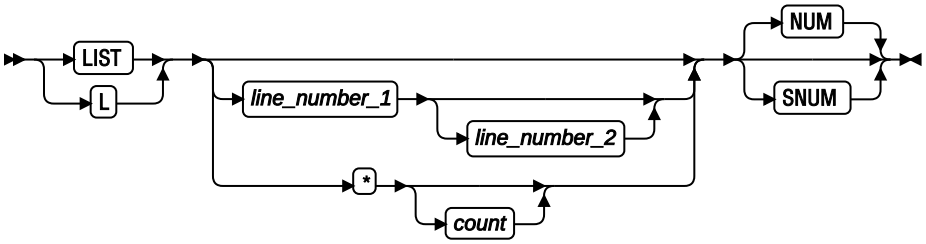
```
*
```

EDIT—LIST subcommand

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

If you do not specify any operands with LIST, the entire data set is displayed.

EDIT—LIST subcommand syntax



EDIT—LIST subcommand operands

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 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.

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

count
specifies the number of lines that you want displayed, starting at the location referred to by the line pointer.

NUM | SNUM

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

SNUM
specifies line numbers are to be suppressed; that is, not displayed at the terminal.

EDIT—LIST subcommand examples

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
```

EDIT—MOVE subcommand

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

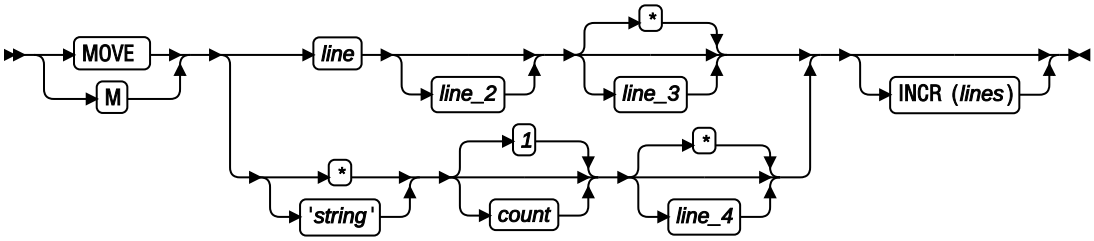
The target line cannot be within the source area, with the exception that the target line (the first line of the target area) can overlap the last line of the source area.

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

If you do not specify any operands with MOVE, the MOVE subcommand is ignored.

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

EDIT—MOVE subcommand syntax



EDIT—MOVE subcommand operands

line_1
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 at the next higher line number.

line_2
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 *line_2*. If *line_2* is not entered, the value defaults to the value of *line_1*; that is, the source becomes one line. Do not enter an asterisk for *line_2*.

If MOVE is followed by two line number operands, the system assumes them to represent *line_1* and *line_3*, and defaults *line_2* to the value of *line_1*.

line_3
specifies the target line number; that is, the line at which the moved_to data area will start. If the *line_3* value corresponds to an existing line, the target line is changed to *line_3* + INCR(lines) and either becomes a new line or displaces an existing line at that location. When 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 *line_3* 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 *line_3* set to *. Note that *line_3* defaults to *.

The value of *line_3* should not fall in the range from *line_1* to *line_2*; that is, the target line must not be in the range being moved. Exception: *Line_3* can be equal to *line_2*.

*****
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.

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

'*string*'
specifies a string of alphanumeric characters with a maximum length equal to or less than the logical record length of the data set you are editing. 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. Enter 1-8 decimal digits, but not zero (0) or asterisk (*). The default for *count* depends on what is specified for '*string*' ('*string*' or *).

If '*string*' is specified and *count* is left blank, the default for *count* is one (1). For example, if you specify:

```
MOVE 'xyz' 99
```

the count default is one (1).

However, if you specify an asterisk (*) for the '*string*', the next operand is treated as the *count* entry. For example, if you specify:

```
MOVE * 99
```

the 99 is treated as the count.

line_4
applies to both numbered and unnumbered data sets. For unnumbered data sets, *line_4* specifies the target line (the line at which the moved-to data area starts) as a relative line number (the 4th line in the data set). For numbered data sets, *line_4* is specified the same as *line_3*. Specifying zero (0) for *line_4* 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 *line_4* set to *. The default for *line_4* is *. However, if '*string*' is specified and *count* is left blank, the operand following '*string*' is treated as the *count* operand and the default for *line_4* (*) is used.

For example, if you specify :

```
MOVE 'xyz' 99
```

the count is 99 and *line_4* is *.

EDIT—MOVE subcommand examples

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: 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 40.
- EDIT supplies an increment of 10.

```
Before:            Enter:                  After:

0010   A             move 60 60 40         0010   A
0020   BB                                   0020   BB
0030   CCC                                  0030   CCC
0040   DDDD                                 0040   DDDD
0050   EEEEE                          CLP   0050   FFFFFF
0060   FFFFFF                               0051   EEEEE
0070   GGGGGGG                              0070   GGGGGGG
0080   HHHHHHHH                             0080   HHHHHHHH
0090   IIIIIIIII                            0090   IIIIIIIII
0100   JJJJJJJJJJ                           0100   JJJJJJJJJJ
0110   KKKKKKKKKKK                          0110   KKKKKKKKKKK
0120   LLLLLLLLLLL                          0120   LLLLLLLLLLL
```

Example 4

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

Known:

- Data set contains nine lines of text.
- Word to be found is men.
- Data set is unnumbered.

```
Before:            Enter:                  After:

NOW IS             top                     NOW IS
THE TIME           move 'men' 1 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.

```
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   LLLLLLLLLLL                                0120   LLLLLLLLLLL
```

Example 6

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

Known:

- Data set contains lines 10 through 120.
- Source lines are 20 through 50.
- Target area starts at line 50.
- EDIT provides an 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   LLLLLLLLLLL                            0120   LLLLLLLLLLL
```

Example 7

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

Known:

- Data set contains lines 10-120.
- Source range is line 70-110.
- Target location is line 50.
- Increment is 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   GGGGGG
0070   GGGGGGG                                    0070   HHHHHHH
0080   HHHHHHHH                                   0080   IIIIIIII
0090   IIIIIIIII                                  0090   JJJJJJJJ
0100   JJJJJJJJJJ                           CLP   0100   KKKKKKKKK
0110   KKKKKKKKKKK                                0101   FFFFFF
0120   LLLLLLLLLLL                                0120   LLLLLLLLLL
```

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   LLLLLLLLLLL                               0120   LLLLLLLLLLL
```

Example 9

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

Known:

- Data set contains lines 0 through 120.
- Line 0 contains data.
- Source lines are 80, 90, and 100.
- Target area starts at line 0.

```
Before:              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   LLLLLLLLLLL                                0160   LLLLLLLLLLL
                    Note: An entry of
                    move 80  100  0  incr(50)
                    produces the results
                    shown at right.  The         0000   ZIP
                    target data is inserted      0050   HHHHHHHH
                    between line 0 and the       0100   IIIIIIIII
                    remainder of the data   CLP  0150   JJJJJJJJJJ
                    set.                         0151   A
                                                 0152   BB
                                                 0153   CCC
                                                 0154   DDDD
                                                 0155   EEEEE
                                                 0156   FFFFFF
                                                 0157   GGGGGGG
                                                 0158   KKKKKKKKKKK
                                                 0159   LLLLLLLLLLL
```

EDIT—PROFILE subcommand

Use the PROFILE subcommand to change the characteristics of your user profile. For a description of the PROFILE command syntax and function, see the “PROFILE command” on page 228.

EDIT—RENUM subcommand

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.

If the data set you are editing contains fixed-length records, new line numbers are placed in the last 8 character positions. There are three exceptions to this general rule:

- Data set type COBOL - first six positions
- Data set type VSBASIC - first five positions

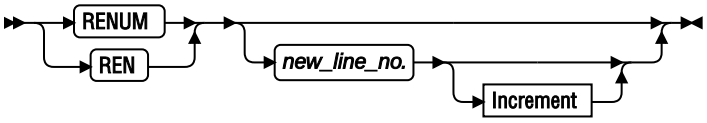
- Data set type ASM and NUM operand specified on EDIT command - positions indicated in NUM operand subfield.

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

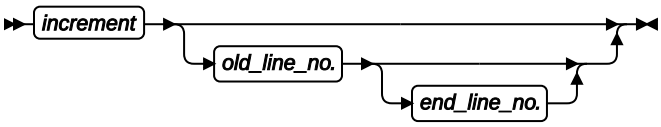
If you are editing variable-length records without sequence numbers, the records are lengthened so that an eight-digit sequence field (five digits if VSBASIC) is prefixed to each record. You are notified if any records have been truncated in the process. Records are truncated when the data length plus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

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

EDIT—RENUM subcommand syntax



Increment



EDIT—RENUM subcommand operands

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 is 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 begins. If this operand is omitted, renumbering starts 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.

EDIT—RENUM subcommand examples

Example 1

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

```
renum
```

Example 2

Operation: Renumber part of a data set with an increment of 1.

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
```

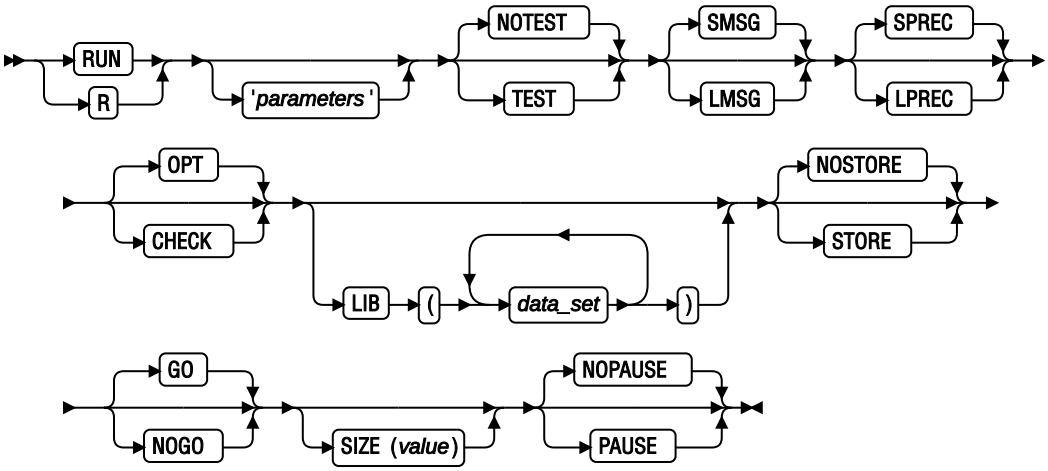
EDIT—RUN subcommand

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 licensed programs. The RUN subcommand selects and invokes the particular licensed program needed to process your source statements.

Any data sets required by your problem program can be allocated before you enter EDIT mode or can be allocated using the ALLOCATE subcommand.

If you want to enter a value for parameters, you should enter this before any of the other keyword operands.

EDIT—RUN subcommand syntax



EDIT—RUN subcommand operands

'*parameters*'
specifies a string of up to 100 characters that is passed to the program that is to be executed. You can specify this operand only for programs that accept parameters.

TEST | NOTEST

TEST
specifies testing is to be performed during execution. This operand is valid for the VSBASIC licensed program only.

NOTEST
specifies no testing is to be done.

If you omit both TEST and NOTEST, the default value is NOTEST.

LMSG | SMSG

LMSG
specifies that you want to receive the longer form of a diagnostic message. This operand is valid for GOFORT statements only.

SMSG
specifies that you want to receive the shorter form of a diagnostic message, if there is one. SMSG is the default.

LPREC | SPREC

LPREC
specifies long precision arithmetic calculations are to be used. This operand is valid for VSBASIC statements only.

SPREC
specifies short precision arithmetic calculations are to be used. SPREC is the default.

CHECK | OPT

CHECK
specifies the PL/I Checkout compiler. This operand is valid for the PL/I licensed program 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 licensed program only.

If both CHECK and OPT are omitted, OPT is the default value for data sets having the PLI descriptive qualifier.

LIB(*data_set*)
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 | NOSTORE

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

GO | NOGO

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

NOGO
specifies 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 area for VSBASIC.

PAUSE | NOPAUSE

PAUSE
specifies that you are 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 you are 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. NOPAUSE is the default.

EDIT—RUN subcommand examples

Example 1

Operation: Start 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 maintain bit patterns.

Known:

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

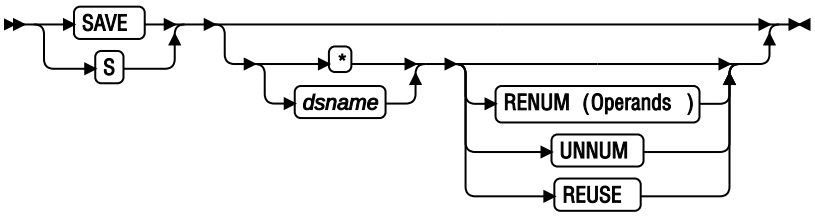
```
run lib(mylib.load)
```

EDIT—SAVE subcommand

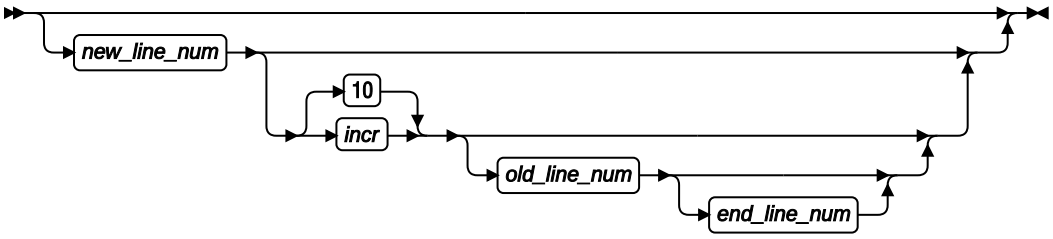
Use the SAVE subcommand to have your data set retained as a permanent data set. If you use SAVE without an operand, the updated version of your data set replaces the original version. When you specify a new data set name as an operand, both the original version and the updated version of the data set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

When you edit a data set with a variable or variable-blocked record format, each record (line) is padded with blanks to the end of the record. When you save the data set, the blanks are eliminated and the length adjusted accordingly.

EDIT—SAVE subcommand syntax



Operands



EDIT—SAVE subcommand operands

*****

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

dsname

specifies a data set name assigned to your edited data set. The new name might be different from the current name. If this operand or an asterisk is omitted, the name entered with the EDIT command is used.

If you specify the name of an existing data set or member of a partitioned data set, that data set or member is replaced by the edited data set. (Before replacement occurs, you are 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) is created with the name you specified. If you specified a member name for a sequentially organized data set, no replacement of the data set takes 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.

REUSE | RENUM | UNNUM

These operands cannot be included unless a data set name or an * is specified.

REUSE

specifies the data set specified in the *dsname* operand is to be reused, if it already exists. You are not prompted for it.

RENUM

specifies the data set is to be renumbered before it is saved.

new_line_number

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

incr

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

old_line_number

specifies the line location within the data set where the renumber process begins. If this operand is omitted, renumbering starts at the beginning of the data set. The *old_line_number* must be equal to or less than the *new_line_number*. If you specify this operand, then you must also specify INCR.

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

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

If the data set you are editing originally contained control characters (ANSI or machine), and you enter SAVE without operands, the following actions apply:

- **For Sequential Data Set:** You are warned that the data set is saved without control characters, that is, the record format is changed. Then you are prompted to enter another data set name for SAVE or a null line to reuse the EDIT data set.
- **For Partitioned Data Set:** Saving into the EDIT data set with a control character attribute is not allowed when it is partitioned. You must save into another data set by specifying a data set name on a subsequent SAVE subcommand entry.

EDIT—SAVE subcommand examples

Example 1

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

Known:

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

```
save index
```

Example 2

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

Known:

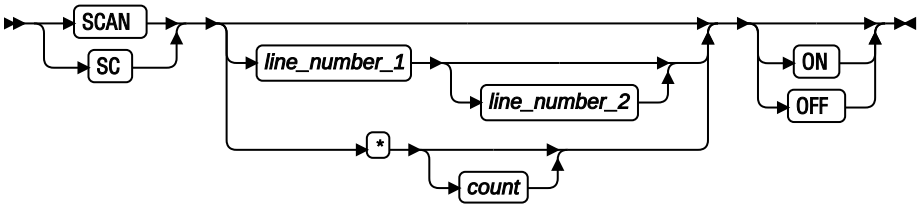
- *new_line_number* 100
- increment(INCR) 50

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

EDIT—SCAN subcommand

Use the SCAN subcommand to request syntax checking services for statements that are processed by the FORTRAN(H) compiler. 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, using the CHANGE subcommand, the INSERT subcommand with the insert data operand, or the insert/replace/delete function.

EDIT—SCAN subcommand syntax



EDIT—SCAN subcommand operands

line_number_1
specifies the number of a line to be checked for proper syntax.

line_number_2
specifies all lines between *line_number_1* and *line_number_2* are to be checked for proper syntax.

*****
specifies 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 | OFF

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

OFF
specifies each line is not to be checked as it is entered in input mode.

If no operands are specified, all existing statements are checked for proper syntax.

EDIT—SCAN Subcommand Examples

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
```

EDIT—SEND subcommand

Use the SEND subcommand to send a message to another terminal user or to the system operator. For a description of the SEND command syntax and function, see the “SEND command” on page 254.

EDIT—SUBMIT subcommand

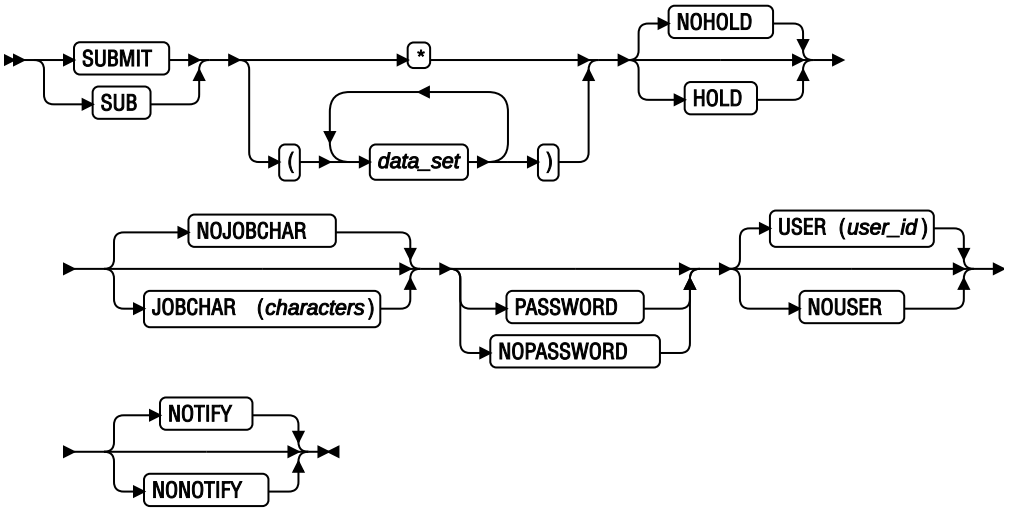
Use the SUBMIT subcommand of EDIT to submit one or more batch jobs for processing. Each job submitted must reside in either a sequential data set, a direct-access data set, or in a member of a partitioned data set. Submitted data sets must be fixed-blocked, 80 byte records. Using the EDIT command to create a CNTL data set provides 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 by a single entry of SUBMIT. Each job must comprise an input job stream (JCL plus data). If the characters in these data sets are lowercase, do not submit data sets with descriptive qualifiers TEXT or PLI.

Job statements are optional. The generated jobname is your user ID plus an identifying character. SUBMIT prompts you for the character and inserts 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 statement(s) being submitted.

You must be authorized by RACF to use SUBMIT.

EDIT—SUBMIT subcommand syntax



EDIT—SUBMIT subcommand operands

(*data_set*)
specifies one or more data set name 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.

*****
specifies the data set you are editing defines the input stream to be submitted. Only the current data set is selected as the input stream. If no operands are entered on the subcommand, the * is the default.

HOLD | NOHOLD

HOLD
specifies SUBMIT is to cause job output to be held for use with the OUTPUT command by defaulting to the held MSGCLASS supplied by the installation manager for the user. If SYSOUT=* or HOLD=YES is specified on the DD statement, then output directed to DD statements is held.

NOHOLD
specifies the job output is not to be held. The default is NOHOLD.

JOBCHAR(*characters*) | NOJOBCHAR

JOBCHAR(*characters*)
specifies characters to be appended to the job name on every JOB statement in the data set being submitted. If you plan to use the STATUS command and your job name is your user ID, use 1 character.

NOJOBCHAR
specifies SUBMIT prompts for job name characters whenever the job name is the user ID. If prompting is not possible, the job name character defaults to the letter X. The default is NOJOBCHAR.

PASSWORD | NOPASSWORD

PASSWORD
indicates a PASSWORD operand is to be inserted on the generated JOB statement by SUBMIT, if RACF is installed. SUBMIT prompts you to enter the password value (in print inhibit mode, if the terminal supports the feature). This operand is not required if the data set has a JOB statement or RACF is not installed. If RACF is installed, then PASSWORD is the default. The password used is:

- The password (if executing in the foreground) entered on the LOGON command initiating the foreground session. The current password is used for RACF-defined users. If you have updated your password using the LOGON command, you must enter the PASSWORD operand with the new password on the SUBMIT command.
- The password on the LOGON command (if executing in the background) in the data set containing the EDIT command. If a LOGON command is not in the data set, the USER and PASSWORD operands are not to be included on the generated JOB statement.

NOPASSWORD
specifies PASSWORD and USER operands are not included on the generated JOB statement. If RACF is not installed, NOPASSWORD is the default.

USER(*user_id*) | NOUSER

USER(*user_id*)
specifies a USER operand is to be inserted on the generated JOB statement, if RACF is installed. The user ID specified is also used as the job name for the generated JOB statement and for job name or user ID comparison for NOJOBCHAR processing (see NOJOBCHAR operand description).

If neither USER or NOUSER is entered and RACF is installed, then USER is the default. The default user ID value used is determined by the following rules. The rules are ordered. If the first rule is met, then the user ID is used.

1. The user ID specified on a LOGON command in the data set containing the EDIT command.
2. The user ID specified on the LOGON command (if executing in the foreground) initiating the foreground session; the user ID specified on the USER operand (if executing in the background - RACF defined users only) on the JOB statement initiating the background session.
3. The default user ID SUBMITJB is used.

NOUSER
specifies generated JOB statements do not include USER and PASSWORD operands. If USER is not specified and RACF is not installed, then NOUSER is the default.

NOTIFY | NONOTIFY

NOTIFY
specifies you are to be notified when your job terminates in the background, if a JOB statement has not been provided. If you do not want to receive messages, the message is placed in the broadcast data set. You must then enter LISTBC to receive the message. If a JOB statement is generated, then NOTIFY is the default.

When you supply your own JOB statement, use the NOTIFY=user_id operand on the JOB statement if you want to be notified when the job terminates. SUBMIT ignores the NOTIFY operand unless it is generating a JOB statement.

NONOTIFY
specifies a termination message is not to be issued or placed in the broadcast data set. The NONOTIFY operand is only recognized when a JOB statement has not been provided with the job that you are processing.

If any of the preceding types of data sets containing two or more jobs is submitted for processing, certain conditions apply:

- The SUBMIT processor builds a job statement for the first job in the first data set, if none is supplied, but does not build job statements 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.
- After 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 statement you supply should have a job name consisting of your user ID and a single identifying character. If the job name is not in this format, you cannot refer to it with the CANCEL command. You are required to specify the job name in the STATUS command if the IBM-supplied exit has not been replaced by your installation and your job name is not your user ID plus a single identifying character.

If you want to provide a job statement, but you also want to be prompted for a unique job name character, put your user ID in the job name field and follow it with blanks so that there is room for SUBMIT to insert the prompted-for character. This allows you to change job names without re-editing the JCL data set.

After SUBMIT has successfully submitted a job for batch processing, it issues a 'jobname(jobid) submitted' message. The job ID is a unique job identifier assigned by the job entry subsystem (JES).

EDIT—SUBMIT subcommand examples

Example 1

Operation: Submit the data set you are editing for batch processing.

Known:

- The data set has no job statement and you do not want to be notified when the job is completed.

```
submit * nonotify
```

EDIT—TABSET subcommand

Use the TABSET subcommand to:

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

Note: The TABSET subcommand is supported only on terminals that support tab setting.

The basic form of the TABSET subcommand causes each strike of the tab key to be translated into blanks corresponding to the column requirements for the data set type. For example, if the name of the data set you are editing has FORT as a descriptive qualifier, the first tabulation setting is in column 7. The values in Table 14 on page 116 are in effect when you first enter the EDIT command.

Table 14: Default tab settings

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

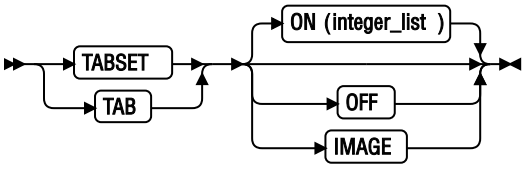
Table 14: Default tab settings (continued)

Data set name descriptive qualifier	Default tab settings columns
DATA	10,20,30,40,50,60
FORT FORTRAN(H) compilers, FORTRAN IV (G1) 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

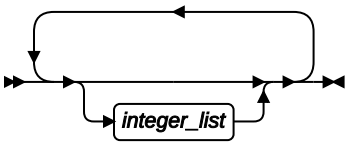
You might find it convenient to have the mechanical tab settings coincide with the logical tab settings. Note that, except for line-numbered COBOL or VSBASIC data sets, the logical tab columns apply only to the data that you actually enter. Because a printed line number prompt is not logically part of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

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

EDIT—TABSET subcommand syntax



integer_list



EDIT—TABSET subcommand operands

ON(*integer_list*)

specifies 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. ON is the default.

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

IMAGE
specifies the next input line defines new tabulation settings. The next line that you type should consist of t's, indicating the column positions of the tab settings, and blanks or any other characters except t. Ten is the maximum number of tab settings 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” on page 118).

EDIT—TABSET subcommand examples

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…
```

EDIT—TOP subcommand

Use the TOP subcommand to change the line pointer in the system to zero, that is, the pointer points 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 points 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.

If the data set is empty, you are notified. However, the current line pointer still takes on a zero value.

EDIT—TOP subcommand syntax

EDIT—TOP subcommand examples

Example 1

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

Known:

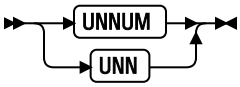
- The data set is not line-numbered.

```
top
```

EDIT—UNNUM subcommand

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

EDIT—UNNUM subcommand syntax



EDIT—UNNUM subcommand examples

Example 1

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

Known:

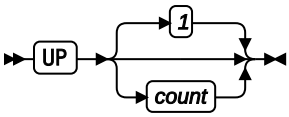
- The data set has line numbers.

```
unnum
```

EDIT—UP subcommand

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 are notified.

EDIT—UP subcommand syntax



EDIT—UP subcommand operands

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 is moved only one line.

EDIT—UP subcommand examples

Example 1

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

```
up
```

Example 2

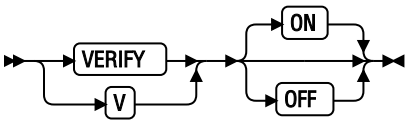
Operation: Change the pointer so that it refers to a line located 17 lines before the location currently referred to.

```
up 17
```

EDIT—VERIFY subcommand

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 do not have verification of changes in the position of the current line pointer.

EDIT—VERIFY subcommand syntax



EDIT—VERIFY subcommand operands

ON
specifies 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. If you omit both ON and OFF, then ON is the default setting.

OFF
specifies you want to discontinue this service.

If the VERIFY subcommand is activated by BOTTOM, CHANGE, COPY, DELETE, DOWN, FIND, MOVE, RENUM, UNNUM and UP, then subcommands change the current line pointer and cause it to be displayed.

EDIT—VERIFY subcommand examples

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

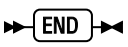
Operation: Terminate the operations of the VERIFY subcommand.

```
verify off
```

END command

Use the END command to end a CLIST. When the system encounters an END command in a CLIST, and the CONTROL MAIN option is not in effect, CLIST execution halts. If the CONTROL MAIN option is in effect, use the EXIT statement to halt the execution of the CLIST. This function is better performed by the EXIT statement.

END command syntax



END command return code

The return code is from the command that executed last.

EXEC command

Use the EXEC command to execute a CLIST or REXX exec.

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

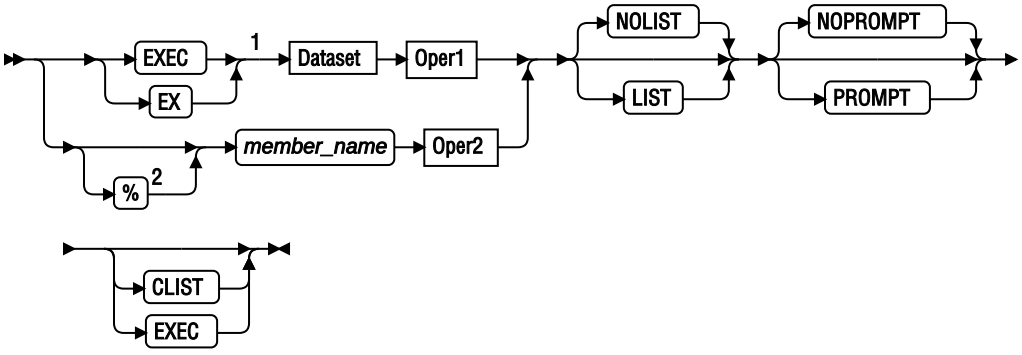
- **Explicit form:** Enter EXEC or EX followed by the name of the data set that contains the CLIST or REXX exec. If you need prompting you should invoke EXEC explicitly with the PROMPT option.
- **Implicit form:** Do *not* enter EXEC or EX; enter only the name of the member to be found in a procedure library such as SYSEXEC or SYSPROC. A procedure library consists of partitioned data sets allocated to the specific file (SYSPROC or SYSEXEC) either dynamically by the ALLOCATE command or as part of the LOGON procedure. TSO/E determines if the member name is a system command before it searches the libraries.
- **Extended implicit form:** Enter a percent sign followed by the member name. TSO/E only searches the procedure library for the specified name. This form is faster because the system doesn't search for commands.

Some of the commands in a CLIST might have symbolic variables for operands. When you specify the EXEC command, you can supply actual values for the system to use in place of the symbolic variables. In addition, when you invoke a REXX exec you can pass arguments on the EXEC command. Specify the arguments in single quotation marks.

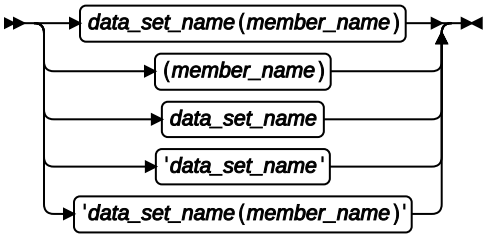
Using EXEC as a subcommand

The EXEC subcommand of EDIT and TEST performs the same basic functions as the EXEC command. However, a CLIST that is executed with an EXEC subcommand can execute only CLIST statements and other subcommands of the EDIT or TEST commands. A REXX exec executed with an EXEC subcommand can execute only REXX statements. When used to execute a REXX exec, the EXEC subcommand can use the data stack to provide information to EDIT or TEST. For information about writing CLISTs, see *z/OS TSO/E CLISTs*. For information about writing REXX execs, see *z/OS TSO/E REXX User's Guide* and *z/OS TSO/E REXX Reference*.

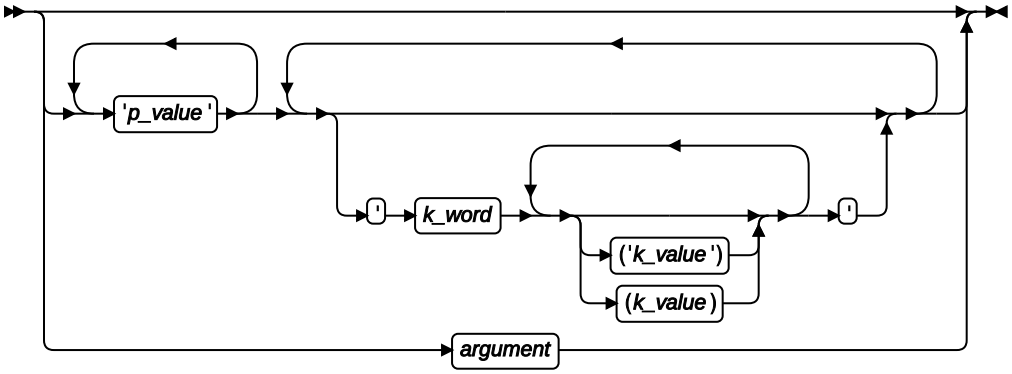
EXEC command syntax



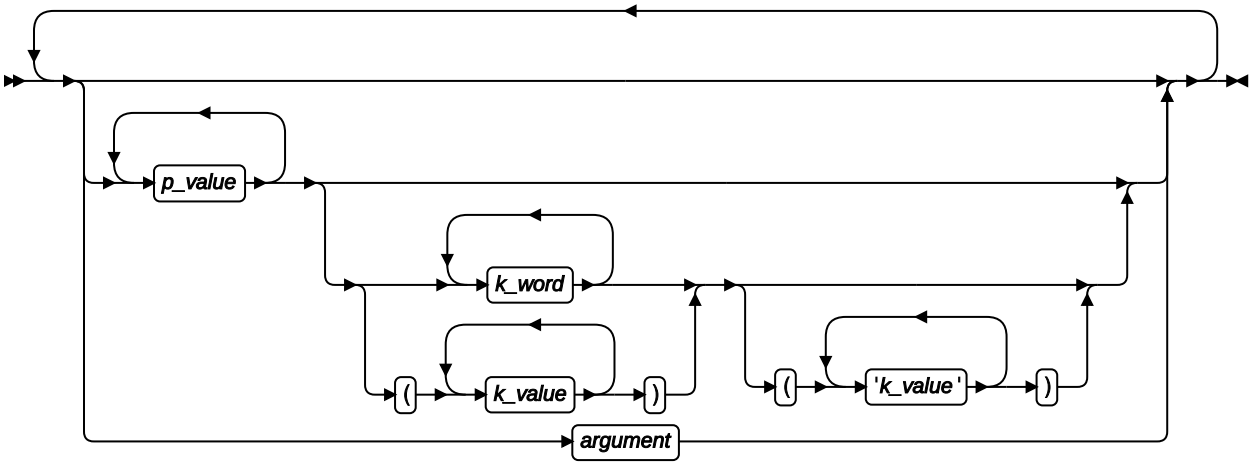
Dataset



Oper1



Oper2



Notes:

[1] The explicit form of the EXEC command.

[2] The implicit (without percent sign) and extended implicit form (with percent sign).

EXEC command operands

***data_set_name*(*member_name*)**
specifies the unqualified name of a partitioned data set whose type is CLIST or exec. The *data_set_name* is the library name such as the name SESSION in the data set PREFIX.SESSION.CLIST.(*member_name*) is the name of the CLIST or exec. For example, to execute `prefix.session.clist(first)`, specify:

```
exec session (first)
```

(*member_name*)
specifies a member of a partitioned data set whose type is CLIST or exec. (*member_name*) is the name of the CLIST or exec. For example, to execute an exec named `prefix.exec(two)`, specify:

```
exec (two) exec
```

data_set_name
specifies the unqualified name of a sequential data set whose type is CLIST or exec. *Data_set_name* is the name of the CLIST or exec. For example, to execute a CLIST named `prefix.test.clist`, specify:

```
exec test.clist
```

'*data_set_name*'
specifies the fully-qualified name of a sequential data set. For example, to execute an exec named `project.num.one`, specify:

```
exec 'project.num.one' exec
```

If the data set is not a sequential, but a partitioned one, a member TEMPNAME is assumed. If such member does not exist, the system will notify you, otherwise it will be executed.

'*data_set_name*(*member_name*)'
specifies the fully-qualified name of a partitioned data set. (*member_name*) is the name of the CLIST or exec. For example, to execute a CLIST named `PROJECT.SPECIAL.$1993(MARCH)`, specify:

```
exec 'PROJECT.SPECIAL.$1993(MARCH)'
```

A CLIST or REXX exec data set may contain line numbers according to the following format:

- Variable blocked — First 8 characters in each record. If the data in columns 1-8 is not numeric, the CLIST or exec treats it as data.
- Fixed blocked — Last 8 characters in each record

You are suggested to use variable blocked records, although you can also use fixed blocked here.

p_value
For use with CLISTs only. A *p_value* is the actual value a user specifies for each positional parameter on the PROC statement. Lowercase values are changed to uppercase.

The user must specify a *p_value* for each positional parameter in the same sequence as each appears on the PROC statement (for example, p_value_1 p_value_2 ... p_value_n).

If a user does not specify a *p_value* for a positional parameter, the CLIST prompts for the value. See "Considerations for passing quotation marks" on page 128 for more information.

argument
For use with execs only. Specifies a parameter passed to an exec.

k_word
For use with CLISTs only. *k_word* is the actual keyword a user specifies. It can be an abbreviation if it is different from all other *k_word* parameters in the EXEC command.

The specification of *k_word* must follow all *p_value* specifications; but *k_words* may be specified in any order.

k_value
A value associated with a *k_word*.

'*k_value*'
k_value is a quoted string. Lowercase values are changed to uppercase.

Specification on the PROC statement: keyword()

- If the user specifies *k_word* without a *k_value*, the CLIST prompts for the value.
- If the user does not specify *k_word*, the associated keyword has a null value.

Specification on the PROC statement: keyword(default value)

- If the user specifies *k_word* without a *k_value*, the CLIST prompts for the value.
- If the user does not specify *k_word*, the CLIST uses the default value.
- If the user specifies *k_word* with a *k_value*, the CLIST uses *k_value*.

See "Considerations for passing quotation marks" on page 128 for more information.

NOLIST | LIST
specifies whether commands and subcommands are to be listed at the terminal as they are executed.

NOLIST
specifies commands and subcommands are not to be listed. The system assumes NOLIST for implicit and explicit EXEC commands. NOLIST is the default.

LIST
specifies commands and subcommands are to be listed. This operand is valid only for the explicit form of EXEC.

NOPROMPT | PROMPT

NOPROMPT
specifies no prompting during the execution of a CLIST or REXX exec. NOPROMPT is the default.

No prompting is allowed during the execution of a program if the NOPROMPT keyword operand of PROFILE has been specified, even if the PROMPT option of EXEC has been specified.

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

The PROMPT keyword is not propagated to nested EXEC commands. If you want to be prompted during execution of the program it invokes, PROMPT must be specified on a nested EXEC command.

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

CLIST | EXEC
specifies whether a CLIST or an exec is to be run. To fully qualify the data set name, the EXEC command adds the suffix CLIST or EXEC to the data set name. For more information about these operands, including what happens when you omit the parameter, see "Using the explicit form of the EXEC command" on page 125.

CLIST
specifies that a CLIST is to be run.

EXEC
specifies that an exec is to be run.

%member_name
specifies the name of a CLIST or exec. If the percent sign (%) is entered, TSO/E searches its procedure libraries for a CLIST or exec only. It does not search for a command. For example, to execute an exec named `prefix.myrexx.exec(new)` that is allocated to a procedure library, specify:

```
%new
```

Suppose the following CLIST 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
```

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 need to specify the implicit form:

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

Note: If the value of a operand is not entered on the EXEC statement, that value is nullified.

Using the explicit form of the EXEC command

Using the explicit form of the EXEC command involves naming the data set that contains the REXX exec or CLIST. You can create the fully-qualified data set name and determine whether it will run as a REXX exec or a CLIST. You can specify either the CLIST or EXEC operand to denote that the data set be run as a REXX exec or CLIST. If you specify neither operand, the data set is run based on the following specifications or defaults:

If you know that the procedure being run is a CLIST, you can code the CLIST operand. If you know that the procedure being run is a REXX exec, you can code the EXEC operand. If you do not code the CLIST or EXEC operand on the EXEC command, the EXEC command processor examines line 1 of the procedure for the characters "REXX" within a comment. (The characters "REXX" can be in uppercase, lowercase, or mixed-case.) This is known as the REXX exec identifier. If the EXEC command finds the REXX exec identifier, the EXEC command runs the procedure as a REXX exec. Otherwise, it runs the procedure as a CLIST.

In addition to determining if a procedure is run as a REXX exec or a CLIST, the CLIST and EXEC operands of the EXEC command determine how to name a non-fully-qualified data set. If you specify EXEC, a non-fully-qualified data set name is suffixed with the "exec" qualifier. If you specify CLIST, or if you omit either EXEC or CLIST, a non-fully-qualified name is suffixed with the qualifier "clist".

The tables that follow show the decision process for a data set that is fully qualified and a data set that is not fully qualified. The outcome of the decision is that the data set will run as either:

- A REXX exec
- A CLIST

Data Set is fully qualified

If you specify:	The procedure runs as a:
EXEC	REXX exec
CLIST	CLIST
Neither and REXX ID is present	REXX exec
Neither and REXX ID is not present	CLIST

Data set name is not fully qualified

If you specify:	Then TSO/E adds:	The procedure runs as a:
EXEC	TSO/E prefix and EXEC suffix	REXX exec
CLIST	TSO/E prefix and CLIST suffix	CLIST
Neither	TSO/E prefix and CLIST suffix and the REXX ID is present	REXX exec
Neither	TSO/E prefix and CLIST suffix and the REXX ID not present	CLIST

The following examples use the explicit form of the EXEC command and show how the procedure runs in each case.

Example 1

Operation: Name is not fully qualified, EXEC or CLIST keyword is specified.

Result:

- The fully-qualified name is prefixed by the PREFIX and is suffixed by "exec" or "clist", unless the non-fully-qualified name already has the appropriate suffix.
- Procedure is run as the keyword specifies.

```
ex tools(mem1) exec
```

runs REXX exec "mem1" from: ‘slk27.tools.exec(mem1)’.

```
ex tools(mem2) clist
```

runs CLIST "mem2" from: ‘slk27.tools.clist(mem2)’.

```
ex tools.exec(mem1) exec
```

runs REXX exec "mem1" from: ‘slk27.tools.exec(mem1)’.

No need to add the "exec" suffix because the name already has the appropriate suffix.

```
ex tools.clist(mem2) clist
```

runs CLIST "mem2" from: ‘slk27.tools.clist(mem2)’.

No need to add the "clist" suffix because the name already has the appropriate suffix.

Example 2

Operation: Name is fully qualified, EXEC or CLIST keyword is specified.

Result:

- Fully-qualified name is as specified
- Procedure is run as the keyword specifies.

```
ex ‘sk127.tools.exec(mem1)’ exec
```

runs REXX exec "mem1" from: ‘slk27.tools.exec(mem1)’

```
ex ‘sk127.tools.clist(mem2)’ clist
```

runs CLIST "mem2" from: ‘slk27.tools.clist(mem2)’.

Example 3

Operation: Name is not fully qualified, EXEC or CLIST keyword is not specified.

Result:

- Fully-qualified name is prefixed by the PREFIX, and is suffixed by "clist", unless the non-fully-qualified name already has the appropriate suffix.
- Procedure is run as a REXX exec if the REXX string is found within a comment in line 1 of the procedure. Otherwise, it is run as a CLIST.

```
ex tools(mem3)
```

runs "mem3" as REXX exec or CLIST depending on what is found in line 1 of procedure "mem3". Whether "mem3" is run as a REXX exec or a CLIST, it is read from: `'slk27.tools.clist(mem3)'`.

```
ex tools.clist(mem3)
```

runs "mem3" as REXX exec or CLIST depending on what is found in line 1 of procedure "mem3". Whether "mem3" is run as a REXX exec or a CLIST, it is read from: `'slk27.tools.clist(mem3)'`.

No need to add the "clist" suffix because the name already has the appropriate suffix.

Example 4

Operation: Name is fully qualified, EXEC or CLIST keyword is not specified.

Result:

- Fully-qualified name is as specified.
- Procedure is run as a REXX exec if the string REXX is found within a comment in line 1 of the procedure. Otherwise, it is run as a CLIST.

```
ex 'slk27.tools exec(mem3)'
```

runs "mem3" as REXX exec or CLIST depending on what is found in line 1 of procedure "mem3".

Using the (extended) implicit form of the EXEC command

When using the implicit form of the EXEC command, TSO/E finds the REXX exec or CLIST as follows.

Table 15 on page 127 lists the search order of the user-, application-, and system-level libraries. Also shown are the ddnames associated with each library level. These ddnames can be allocated either dynamically by the ALLOCATE command or included as part of a logon procedure.

Table 15: Library search order

Search order	Library level		Associated ddname
1.	User	REXX exec	SYSUEXEC
2.	User	CLIST	SYSUPROC
3.	Application	REXX exec	Define with FILE or DATASET operand
4.	Application	CLIST	Define with FILE or DATASET operand
5.	System	REXX exec	SYSEXEC (installation can define this ddname)
6.	System	CLIST	SYSPROC

With the default settings that TSO/E provides, and before an ALTLIB command is invoked, TSO/E searches the system EXEC library (default ddname SYSEXEC) first, followed by the system CLIST library (ddname SYSPROC). Note that your system programmer can change this by

- Defining an alternate ddname of SYSEXEC

- Indicating that TSO/E is not to search the system-level exec ddname of SYSEXEC. Then only the system-level CLIST (SYSPROC) is searched.

You can alter the default library search order by using either the ALTLIB command or the EXECUTIL command.

- Use EXECUTIL to indicate that the system-level exec ddname is to be searched for the duration of the current REXX language processor environment.
- Use ALTLIB to indicate that the system-level exec ddname is to be searched for the duration of the current application. ALTLIB always overrides EXECUTIL within an application.

Use ALTLIB DISPLAY to see which libraries are being searched for.

The following example uses the implicit form of the EXEC command. It shows how the procedure is run. In this example, assume that the TSO/E prefix is ‘slk27’.

Example 1

Operation: Run an implicit procedure.

Result:

- If the implicit procedure was found in the data set allocated to the SYSEXEC file, it is run as a REXX exec.
- If the implicit procedure was found in the data set allocated to the SYSPROC file, it is run as a REXX exec if the string REXX appears in a comment on line 1 of the procedure. Otherwise, it is run as a CLIST.

```
%mem4
```

runs "mem4" as REXX exec, if "mem4" was found in SYSEXEC or runs "mem4" as REXX exec or CLIST, depending on what is found in line 1 of procedure "mem4", if "mem4" was found in SYSPROC.

Considerations for passing quotation marks

Considerations for specifying parameters that contain single quotation marks (apostrophes):

- implicit invocation - specify the exact string.
- explicit invocation - specify two apostrophes for each apostrophe within the string. For example, to pass the string "`It's`" specify:

  ```
  It''s
  ```

  To pass the three-parameter string "`It's 2 o'clock`" specify:

  ```
  It''s 2 o''clock
  ```

Considerations for specifying parameters that are quoted strings:

- implicit invocation:
  - *p_value* - specify the exact string. For example, to pass the fully-qualified data set name `'USER33.MASTER.BACKUP'` specify:

    ```
    'user33.master.backup'
    ```

  - *k_word*('*k_value*') - to pass the same fully-qualified data set name as shown in the previous example as a *k_value*, specify:

    ```
    dsn(''''user33.master.backup'''')
    ```

- explicit invocation:

– *p_value* - specify two quotation marks for each enclosing quote. For example, to pass the fully-qualified data set name `'USER33.MASTER.BACKUP'` specify:

```
'''user33.master.backup'''
```

The outermost set of quotation marks is required as part of the syntax.

– *k_word*('*k_value*') - to pass the same fully-qualified data set name as shown in the previous example as a *k_value*, specify

```
'dsn(''''''user33.master.backup'''''')'
```

The number of enclosing quotation marks must be doubled because the entire specification is itself a quoted string.

EXEC command return codes

Table 16 on page 129 lists the return codes of EXEC command.

Table 16: EXEC command return codes

Return codes	Meaning
0	Processing successful.
12	Processing unsuccessful.
Other	Return code is from the EXEC command exit routines or from the REXX exec that was executed.

If your installation uses EXEC command exit routines and those routines indicate that the reason code is used as the return code from EXEC, you may receive return codes other than those listed. For more information about reason and return codes from EXEC, see *z/OS TSO/E Customization*.

EXEC command examples

Example 1

Operation: Start a CLIST using the explicit form of EXEC.

Known:

- The name of the data set that contains the CLIST is SLK27.USER.CLIST(MEMBER)
- The user's TSO/E prefix is SLK27.

```
ex 'slk27.user.clist(member)'
```

Example 2

Operation: Start a CLIST to invoke the assembler.

Known:

- The name of the data set that contains the CLIST is RBJ21.FASM.CLIST.
- The CLIST consists of:

```
PROC 1 NAME
 free file(sysin,sysprint)
 delete (&name..list,&name..obj)
 allocate dataset(&name...asm) file(sysin) SHR  keep
 allocate dataset(&name..list) file(sysprint) -
  block(132) space(300,100)
 allocate dataset(&name..obj) file(syspunch) block(80) -
  space(100,50)
 allocate file(sysut1) space(3,1) cylinders new delete
```

```
  allocate file(sysut2) space(3,1) cylinders new delete
  allocate file(sysut3) space(3,1) cylinders new delete
  allocate file(syslib) da('d82ljp1.tso.macro',
   'sys1.maclib') shr
  call '*(ASMA90)' 'deck,noobj,rent'
  free file(sysut1,sysin,sysprint, -
   syspunch,syslib)
  allocate file(sysin) da(*)
  allocate file(sysprint) da(*)
```

Note: You can use a period to delimit a symbolic variable. However, follow the first period with another period. The first period is the delimiter that is removed during symbolic substitution of the variable. The second period remains unchanged.

- The module to be assembled is TGETASIS.
- You want to have the names of the commands in the CLIST displayed at your terminal as they are executed.

To execute the CLIST, enter:

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

The display at your terminal need to be similar to:

```
EX FASM 'TGETASIS' LIST
FREE FILE(SYSIN,SYSPRINT)
DELETE (TGETASIS.LIST,TGETASIS.OBJ)
IDC0550I ENTRY (A) D82LJP1.TGETASIS.LIST DELETED
IDC0550I ENTRY (A) D82LJP1.TGETASIS.OBJ DELETED
ALLOCATE DATASET(TGETASIS.ASM) FILE(SYSIN) OLD KEEP
ALLOCATE DATASET(TGETASIS.LIST) FILE(SYSPRINT)
 BLOCK(132) SPACE(300,100)
ALLOCATE DATASET(TGETASIS.OBJ) FILE(SYSPUNCH)
 BLOCK(80) SPACE(100,50)
ALLOCATE FILE(SYSUT1) SPACE(3,1) CYLINDERS NEW DELETE
ALLOCATE FILE(SYSUT2) SPACE(3,1) CYLINDERS NEW DELETE
ALLOCATE FILE(SYSUT3) SPACE(3,1) CYLINDERS NEW DELETE
ALLOCATE FILE(SYSLIB) DA('D82LJP1.TSO.MACRO',
 'SYS1.MACLIB') SHR
CALL '*(ASMA90)' 'DECK,NOOBJ,RENT'
FREE FILE(SYSUT1,SYSUT2,SYSUT3,SYSIN,SYSPRINT,
 SYSPUNCH,SYSLIB)
ALLOCATE FILE(SYSIN) DA(*)
ALLOCATE FILE(SYSPRINT) DA(*)
READY
```

Example 3

Operation: Assume that the CLIST in Example 2 has been stored in a CLIST library, which was allocated to the SYSPROC file ID. Run the CLIST using the implicit form of EXEC.

Known:

- The name of the member of the partitioned data set
- that contains the CLIST is FASM2.

```
fasm2 tgetasis
```

Example 4

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

Known:

- The CLIST named SWITCH is contained in a CLIST library named MASTER.CLIST which is allocated as SYSPROC.
- The CLIST consists of:

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

```
RENAME &DSN2 &DSN1
RENAME TEMPSAVE &DSN2
```

If you have a user ID of USER33 and you want to switch the names of two data sets MASTER.BACKUP and USER33.GOODCOPY, you can invoke the CLIST 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 you use the implicit form, the specification of quoted strings in the value list is made simpler because the value list itself is not a quoted string.

Example 5

Operation: Start a REXX exec using the explicit form of EXEC.

Known:

- The name of the data set that contains the REXX exec is LMW18.USER.EXEC(MEMBER)
- The user's TSO/E prefix is LMW18.

```
ex 'lmw18.user.exec(member)' exec
```

Note that the exec operand used in this example is optional. When a fully quoted data set name is specified, the exec operand (or CLIST operand when executing a CLIST) is not required.

Example 6

Operation: Assume that the REXX exec in Example 5 has been stored in a REXX library, which was allocated to the SYSEXEC file ID. Run the REXX exec using the implicit form of EXEC.

Known:

- The name of the member of the partitioned data set that contains the REXX exec is MEMBER.

```
member
```

Example 7

Operation: Enter a fully-qualified data set name as an argument in an explicitly executed REXX exec.

Known:

- The REXX exec named SWITCH is contained in a REXX library named MASTER.EXEC which is allocated to SYSPROC.
- The REXX exec consists of:

```
PARSE ARG dsn1 dsn2
'RENAME' dsn1 'TEMPSAVE'
'RENAME' dsn2  dsn1
'RENAME TEMPSAVE' dsn2
```

If you have a user ID of USER33 and you want to switch the names of two data sets MASTER.BACKUP and USER33.GOODCOPY, you can invoke the REXX exec as follows:

Explicit form:

```
exec 'master.exec(switch)' '''master.backup'' goodcopy' exec
```

Extended implicit form:

```
%switch 'master.backup' goodcopy
```

Note that when you use the implicit form, the specification of quoted strings in the value list is made simpler because the value list itself is not a quoted string.

Example 8

Operation: Pass an argument string containing values separated by commas to a REXX exec.

Known:

- The REXX exec named GETARG is contained in a REXX library named REXX.EXEC, which is allocated to file SYSEXEC.
- The REXX exec consists of:

```
PARSE ARG A ',' B
SAY 'Value of A is:' A
SAY 'Value of B is:' B
```

Implicit form:

```
GETARG 1,2
```

Extended implicit form:

```
%GETARG 1,2
```

Explicit form:

```
ex 'REXX.EXEC(GETARG)' '1,2'
```

Note: If you want to pass an argument string that contains values separated by commas and the first value is null (that is, the argument string begins with a comma), then the explicit form must be used.

For example, to pass the argument string ",3" to the GETARG exec, you must specify:

```
ex 'REXX.EXEC(GETARG)' ',3'
```

In this case, GETARG is passed the two character argument string ",3". The PARSE ARG A ',' B instruction parses the argument string to obtain a null value for A, and a value of 3 for B.

If an implicit invocation is used, the leading comma is stripped from the argument string passed to the exec. That is,

```
GETARG ,3
```

results in the 1 character string "3" being passed to the exec.

EXECUTIL command

The EXECUTIL command is a TSO/E REXX command that lets you change various characteristics that control how an exec executes in the TSO/E address space. You can use EXECUTIL:

- In an exec
- From TSO/E READY mode
- From ISPF - the ISPF command line or ISPF option 6 (enter a TSO/E command or CLIST)
- In a CLIST. You can use EXECUTIL in a CLIST to affect exec processing. However, it has no effect on CLIST processing.

You can also use EXECUTIL with the HI, HT, RT, TS, and TE operands from a program that is written in a high-level programming language by using the TSO/E service facility. From READY mode or ISPF, the HI, HT, and RT operands are not applicable because an exec is not currently executing.

Use EXECUTIL to:

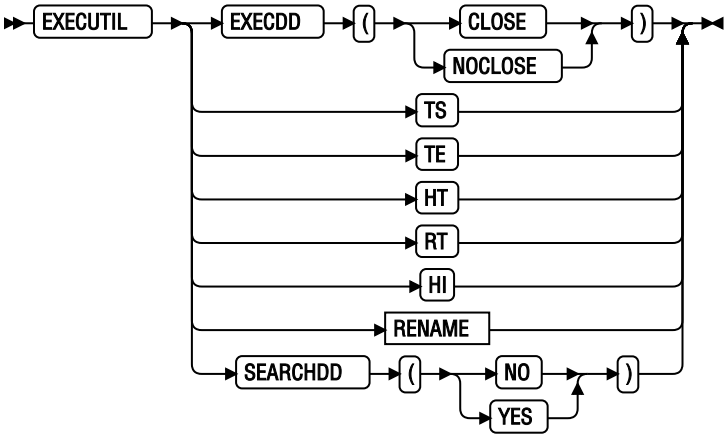
- Specify whether the system exec library, whose default name is SYSEXEC, is to be closed upon completion of the exec or is to remain open
- Start and stop tracing of an exec
- Stop the execution of an exec
- Suppress and resume terminal output from an exec
- Change entries in a function package directory
- Specify whether the system exec library (the default is SYSEXEC) is to be searched in addition to SYSPROC.

Additional considerations for using EXECUTIL

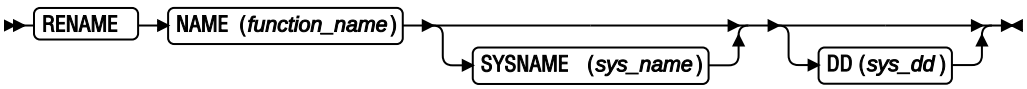
- All of the EXECUTIL operands are mutually exclusive, that is, you can only specify one of the operands on the command.
- The HI, HT, RT, TS, and TE operands on the EXECUTIL command are also, by themselves, *immediate commands*. Immediate commands are commands that can be issued from the terminal if an exec is executing and you press the attention interrupt key and enter attention mode. These commands are processed immediately. *z/OS TSO/E REXX Reference*, describes the immediate commands.
- In general, EXECUTIL works on a language processor environment basis. That is, EXECUTIL only affects the current environment in which EXECUTIL is issued. For example, if you are in split screen in ISPF and issue EXECUTIL TS from the second ISPF screen to start tracing, only execs that are invoked from that ISPF screen are traced. If you invoke an exec from the first ISPF screen, the exec is not traced.

  Using the EXECDD and SEARCHDD operands may affect subsequent language processor environments that are created. *z/OS TSO/E REXX Reference*, describes the concept of language processor environments and how EXECUTIL EXECDD and EXECUTIL SEARCHDD may affect more than one environment.

EXECUTIL command syntax



RENAME



EXECUTIL command operands

EXECDD(CLOSE | NOCLOSE)
Specifies whether the system exec library is to be closed upon completion of the exec.

CLOSE
causes the system exec library, whose default name is SYSEXEC, to be closed upon completion of the exec. This condition can be changed by issuing the EXECUTIL EXECDD(NOCLOSE) command.

NOCLOSE
causes the system exec library to remain open upon completion of the exec. This is the default condition and can be changed by issuing the EXECUTIL EXECDD(CLOSE) command. The selected option remains in effect until it is changed by the appropriate EXECUTIL command, or until the current environment is terminated.

The EXECDD operand affects the ddname specified in the LOADDD field in the module name table. The default is SYSEXEC. *z/OS TSO/E REXX Reference*, describes the module name table in detail.

Any libraries defined using the ALTLIB command are not affected by the EXECDD operand. SYSPROC is also not affected.

Note: Specify EXECDD(CLOSE) or EXECDD(NOCLOSE) before running any execs out of the SYSEXEC file. If you attempt to use EXECDD(CLOSE) or EXECDD(NOCLOSE) after SYSEXEC has been opened, you might not get the desired result because the SYSEXEC file must be closed at the same MVS task level at which it was opened.

TS
Use TS (Trace Start) to start tracing execs. Tracing lets you interactively control the execution of an exec and debug problems. For more information about the interactive debug facility, see *z/OS TSO/E REXX Reference*.

If you issue EXECUTIL TS from READY mode or ISPF, tracing is started for the next exec you invoke. Tracing is then in effect for that exec and any other execs it calls. Tracing stops:

- When the original exec completes.
- If one of the invoked execs specifies EXECUTIL TE.
- If one of the invoked execs calls a CLIST, which specifies EXECUTIL TE.
- If you enter attention mode while an exec is executing and issue the TE immediate command.

If you use EXECUTIL TS in an exec, tracing is started for all execs that are executing. This includes the current exec that contains EXECUTIL TS, any execs it invokes, and any execs that were executing when the current exec was invoked. Tracing remains active until all currently executing execs complete or an exec or CLIST contains EXECUTIL TE.

For example, suppose exec A calls exec B, which then calls exec C. If exec B contains the EXECUTIL TS command, tracing is started for exec B and remains in effect for both exec C and exec A. Tracing stops when exec A completes. However, if one of the execs contains EXECUTIL TE, tracing stops for all of the execs.

If you use EXECUTIL TS in a CLIST, tracing is started for all execs that are executing, that is, for any exec the CLIST invokes or execs that were executing when the CLIST was invoked. Tracing stops when the CLIST and all currently executing execs complete or if an exec or CLIST contains EXECUTIL TE. For example, suppose an exec calls a CLIST and the CLIST contains the EXECUTIL TS command. When control returns to the exec that invoked the CLIST, that exec is traced.

You can use EXECUTIL TS from a program by using the TSO/E service facility. For example, suppose an exec calls a program and the program encounters an error. The program can invoke EXECUTIL TS using the TSO/E service facility to start tracing all execs that are currently executing.

You can also press the attention interrupt key, enter attention mode, and then enter TS to start tracing or TE to stop tracing. *z/OS TSO/E REXX Reference*, describes the TS and TE immediate commands.

TE

Use TE (Trace End) to end tracing execs. The TE operand is not applicable in READY mode because an exec is not currently running. However, if you issued EXECUTIL TS to trace the next exec you invoke and then issued EXECUTIL TE, the next exec you invoke is not traced.

If you use EXECUTIL TE in an exec or CLIST, tracing is ended for all execs that are currently running. This includes execs that were executing when the exec or CLIST was invoked and execs that the exec or CLIST calls. For example, suppose exec A calls CLIST B, which then calls exec C. If tracing was on and CLIST B contains EXECUTIL TE, tracing is stopped and execs C and A are not traced.

You can use EXECUTIL TE from a program by using the TSO/E service facility. For example, suppose tracing has been started and an exec calls a program. The program can invoke EXECUTIL TE using the TSO/E service facility to stop tracing of all execs that are currently executing.

You can also press the attention interrupt key, enter attention mode, and then enter TE to stop tracing. *z/OS TSO/E REXX Reference*, describes the TE immediate command.

HT

Use HT (Halt Typing) to suppress terminal output generated by an exec. The exec continues executing. HT suppresses any output generated by REXX instructions or functions (for example, the SAY instruction) and REXX informational messages. REXX error messages are still displayed. Normal terminal output resumes when the exec completes. You can also use EXECUTIL RT to resume terminal output.

HT has no effect on CLISTs or commands. If an exec invokes a CLIST and the CLIST generates terminal output, the output is displayed. If an exec invokes a command, the command displays messages.

Use the HT operand in either an exec or CLIST. You can also use EXECUTIL HT from a program by using the TSO/E service facility. If the program invokes EXECUTIL HT, terminal output from all execs that are currently executing is suppressed. EXECUTIL HT is not applicable from READY mode or ISPF because no execs are currently executing.

If you use EXECUTIL HT in an exec, output is suppressed for all execs that are executing. This includes the current exec that contains EXECUTIL HT, any execs the exec invokes, and any execs that were executing when the current exec was invoked. Output is suppressed until all currently executing execs complete or an exec or CLIST contains EXECUTIL RT.

If you use EXECUTIL HT in a CLIST, output is suppressed for all execs that are executing, that is, for any exec the CLIST invokes or execs that were executing when the CLIST was invoked. Terminal output resumes when the CLIST and all currently executing execs complete or if an exec or CLIST contains EXECUTIL RT.

For example, suppose exec A calls CLIST B, which then calls exec C. If the CLIST contains EXECUTIL HT, output is suppressed for both exec A and exec C.

If you use EXECUTIL HT and want to display terminal output using the SAY instruction, you must use EXECUTIL RT before the SAY instruction to resume terminal output.

RT

Use RT (Resume Typing) to resume terminal output that was previously suppressed. Use the RT operand in either an exec or CLIST. You can also use EXECUTIL RT from a program by using the TSO/E service facility. If the program invokes EXECUTIL RT, terminal output from all execs that are currently executing is resumed. EXECUTIL RT is not applicable from READY mode or ISPF because no execs are currently executing.

If you use EXECUTIL RT in an exec or CLIST, typing is resumed for all execs that are executing.

HI

Use HI (Halt Interpretation) to halt the interpretation of all execs that are currently running in the language processor environment. From either an exec or a CLIST, EXECUTIL HI halts the interpretation of all execs that are currently running. If an exec calls a CLIST and the CLIST contains EXECUTIL HI, the exec that invoked the CLIST stops processing.

EXECUTIL HI is not applicable from READY mode or ISPF because no execs are currently executing.

You can use EXECUTIL HI from a program by using the TSO/E service facility. If the program invokes EXECUTIL HI, the interpretation of all execs that are currently running is halted.

If an exec enables the halt condition trap and the exec includes the EXECUTIL HI command, the interpretation of the current exec and all execs the current exec invokes is halted. However, any execs that were executing when the current exec was invoked are not halted. These execs continue executing. For example, suppose exec A calls exec B, exec B specifies EXECUTIL HI and also contains a SIGNAL ON HALT instruction (with a HALT: label). When EXECUTIL HI is processed, control is given to the HALT subroutine. When the subroutine completes, exec A continues executing at the statement that follows the call to exec B. For more information about the SIGNAL instruction, see *z/OS TSO/E REXX Reference*.

RENAME

Use EXECUTIL RENAME to change entries in a function package directory. A function package directory contains information about the functions and subroutines that make up a function package. *z/OS TSO/E REXX Reference*, describes function packages and a function package directory.

A function package directory contains the following fields for each function and subroutine:

- Function_name -- the name of the external function or subroutine that is used in an exec.
- Addr -- the address, in storage, of the entry point of the function or subroutine code.
- Sys_name -- the name of the entry point in a load module that corresponds to the code that is called for the function or subroutine.
- Sys_dd -- the name of the DD from which the function or subroutine code is loaded.

You can use EXECUTIL RENAME with the SYSNAME and DD operands to change an entry in a function package directory as follows:

- Use the SYSNAME operand to change the *sys_name* of the function or subroutine in the function package directory. When an exec invokes the function or subroutine, the routine with the new *sys_name* is invoked.
- Use EXECUTIL RENAME NAME(*function_name*) without the SYSNAME and DD operands to flag the directory entry as null. This causes the search for the function or subroutine to continue because a null entry is bypassed. The system will then search for a load module and an exec or both. *z/OS TSO/E REXX Reference*, describes the complete search order.

EXECUTIL RENAME clears the *addr* field in the function package directory to X'00'. When you change an entry, the name of the external function or subroutine is not changed, but the code that the function or subroutine invokes is replaced.

You can use EXECUTIL RENAME to change an entry so that different code is used.

NAME(*function_name*)
specifies the name of the external function or subroutine that is used in an exec. This is also the name in the *function_name* field in the directory entry.

SYSNAME(*sys_name*)
specifies the name of the entry point in a load module that corresponds to the package code that is called for the function or subroutine. If SYSNAME is omitted, the *sys_name* field in the package directory is set to blanks.

DD(*sys_dd*)
specifies the name of the DD from which the package code is loaded. If DD is omitted, the *sys_dd* field in the package directory is set to blanks.

SEARCHDD(YES | NO)
specifies whether the system exec library (the default is SYSEXEC) should be searched when execs are implicitly invoked.

YES
indicates that the system exec library (SYSEXEC) is searched, and if the exec is not found, SYSPROC is then searched.

NO
indicates that SYSPROC only is searched.

EXECUTIL SEARCHDD lets you dynamically change the search order.

Note: EXECUTIL SEARCHDD generally affects the current language processor environment in which it is invoked. If you use EXECUTIL SEARCHDD from TSO/E READY mode, when you invoke ISPF, the new search order may also be in effect for ISPF. This depends on the values your installation uses for the initialization of a language processor environment. For more information about how the search order is defined and how it can be changed, see *z/OS TSO/E REXX Reference*.

ALTLIB affects how EXECUTIL operates to determine the search order. If you use ALTLIB to indicate that user-level, application-level, or system-level libraries are to be searched, ALTLIB operates on an application basis. For more information, see "ALTLIB command" on page 47.

If you use EXECUTIL SEARCHDD, the new search order remains in effect until you issue EXECUTIL SEARCHDD again, the language processor environment terminates, or you use ALTLIB.

EXECUTIL command return codes

Table 17 on page 137 lists the return codes of EXECUTIL command.

Table 17: EXECUTIL command return codes

Return codes	Meaning
0	Processing successful.
12	Processing unsuccessful. An error message has been issued.

EXECUTIL command examples

Example 1

Operation: Your installation uses both SYSEXEC and SYSPROC to store execs and CLISTs. All of the execs you work with are stored in SYSEXEC and your CLISTs are stored in SYSPROC. Currently, your system searches SYSEXEC and SYSPROC and you do not use ALTLIB.

You want to work with CLISTs only and do not need to search SYSEXEC. To change the search order and have the system search SYSPROC only, use the following command:

```
EXECUTIL SEARCHDD(NO)
```

Example 2

Operation: You are updating an exec and including a new internal subroutine. You want to trace the subroutine to test for any problems. In your exec, include EXECUTIL TS at the beginning of your subroutine and EXECUTIL TE when the subroutine returns control to the main program. For example:

```
/*  REXX program  */
MAINRTN:
⋮
CALL SUBRTN
EXECUTIL TE
⋮
EXIT
/*  Subroutine follows  */
SUBRTN:
EXECUTIL TS
⋮
RETURN
```

Example 3

Operation: You want to invoke an exec and trace it. The exec does not contain EXECUTIL TS or the TRACE instruction. Instead of editing the exec and including EXECUTIL TS or a TRACE instruction, you can enter the following from READY mode:

```
EXECUTIL TS
```

When you invoke the exec, the exec is traced. When the exec completes executing, tracing is off.

Example 4

Operation: Suppose an external function called PARTIAL is part of a function package. You have written your own function called PARTIAL or a new version of the external function PARTIAL and want to execute your new PARTIAL function instead of the one in the function package. Your new PARTIAL function may be an exec or may be stored in a load module. You must flag the entry for the PARTIAL function in the function package directory as null in order for the search to continue to execute your new PARTIAL function. To flag the PARTIAL entry in the function package directory as null, use the following command:

```
EXECUTIL RENAME NAME(PARTIAL)
```

When you execute the function PARTIAL, the null entry for PARTIAL in the function package directory is bypassed. The system will continue to search for a load module and exec or both that is called PARTIAL.

FREE command

Use the FREE command to release (deallocate) previously allocated data sets or UNIX file system files 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 can be allocated to you at any one time. The allowable number must be large enough to accommodate:

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

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

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

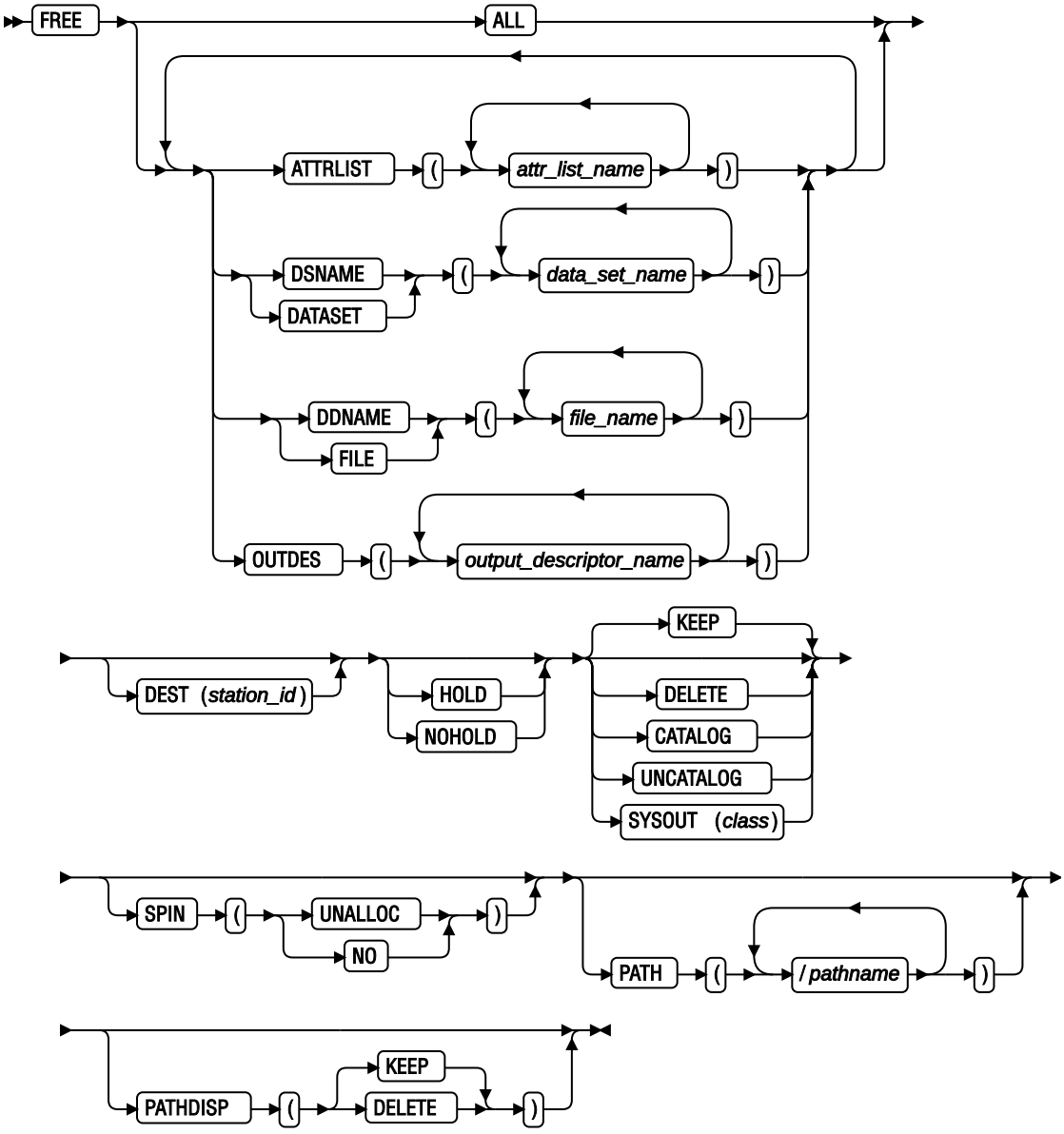
When you free SYSOUT data sets, you can change their output class to make them available for processing by an output writer, or route them to another user.

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.

UNALLOC is the alias of FREE and is intended for use under TEST because FREE is an alias for the FREEMAIN subcommand.

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 command syntax



Note: DELETE is the only disposition that is valid for SYSOUT data sets.

FREE command operands

ALL
requests deallocation of all dynamically allocated data sets, files, and attribute lists that are not marked in-use.

DSNAME(*data_set_name*) | DATASET(*data_set_name*)
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 can contain a member name in parentheses. If you omit this operand, you must specify either FILE, DDNAME, or the ATTRLIST operand.

DDNAME(*file_name*) | FILE(*file_name*)
specifies one or more file names that identify the data sets to be freed. If you omit this operand, you must specify either the DATASET or DSNAME or the ATTRLIST operand.

ATTRLIST(*attr_list_names*)
specifies the names of one or more attribute lists that you want to delete. If you omit this operand, you must specify either the DATASET or DSNAME or the FILE or DDNAME operand.

DEST(*station_id*)
specifies a name of a remote workstation to which the SYSOUT data sets are directed when ready for deallocation. The station ID is a 1 to 8 character name. If this operand is omitted on the FREE command for SYSOUT data sets, the data sets are directed to the workstation specified at the time of allocation.

HOLD | NOHOLD

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

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

KEEP | DELETE | CATALOG | UNCATALOG | SYSOUT(*class*)

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

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

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

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

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 are assigned to this class and placed in the output queue for processing by an output writer. To free a file to SYSOUT, the file must have previously been allocated to SYSOUT.

The changed SYSOUT class characteristics are used in processing the output with the exception of the spool space allocation attribute. The spool space allocation for the SYSOUT data set is unchanged from what was specified at data set allocation time, either through SYSOUT class definition in JES or through the dynamic allocation parameters

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. It can also be freed by entering FREE ALL.

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

OUTDES(*output_descriptor_name*)
specifies a list of output descriptor names, previously defined by the OUTDES command, that are to be freed. Only output descriptors defined by the OUTDES command are freed. You cannot free output descriptors defined in the LOGON procedure.

For more information about the OUTDES command, see the "OUTDES command" on page 188.

SPIN(UNALLOC | NO)
specifies when the system should make the SYSOUT data set available for printing.

UNALLOC
specifies that the system should make the SYSOUT data set available for printing immediately after deallocation.

NO
specifies that the system should make the SYSOUT data set available for printing at the end of the step.

Note:

1. If the SPIN keyword is not specified, FREE does not change the SPIN value of the SYSOUT data set.
2. When the SPIN keyword is specified, you must also specify UNALLOC or NO. If you specify a parameter that is not UNALLOC or NO, or the parameter is missing, FREE will prompt you to specify the parameter.
3. The SPIN keyword specified on the FREE command overrides the SPIN keyword specified on the ALLOCATE command.
4. If the SEGMENT keyword is specified on the ALLOCATE command, the system prints the SYSOUT data set regardless of the SPIN specification on either the ALLOCATE command or FREE command.

PATH(/*pathname*)
identifies a UNIX file system file.

A pathname consists of the names of the directories from the root to the file being identified, and the name of the file. The form is `/name1/name2/…/namen`.

A pathname begins with a slash (/). The system treats any consecutive slashes like a single slash. The pathname can be 2 to 250 characters, including the slash.

Values for pathname consist of printable characters from X'40' to X'FE'. Enclose the pathname in apostrophes if it contains any character other than the following characters:

```
Upper case letters                Numbers
Special characters (#,$, or @)    Slash (/)
Asterisk (*)                      Plus (+)
Hyphen (-)                        Period (.)
Ampersand (&)
```

A pathname is case sensitive. Thus, `'/usr/joe'` and `/usr/J0E` define two different files.

PATHDISP(KEEP | DELETE)
modifies the disposition of a UNIX file as part of DEALLOCATION or FREE processing.

KEEP
specifies that the file should be kept after processing.

DELETE
specifies that the file should be deleted after processing.

FREE command return codes

Table 18 on page 141 lists the return codes of FREE command.

Table 18: FREE command return codes

Return codes	Meaning
0	Processing successful.

Table 18: FREE command return codes (continued)	
Return codes	**Meaning**
12	One of the following occurred: • Processing unsuccessful. An error message was issued. • The file or data set was deallocated, but the disposition specified on the FREE command was overridden by the disposition of the file or data set. An informational message was issued.

FREE command examples

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 file names (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(dsbparms attribut)
```

Example 5

Operation: Free all dynamically allocated data sets, files, and attribute lists.

```
free all
```

Example 6

Operation: Free a file and the dynamic output descriptor.

Known:

- The name of the file: SYSPRINT
- The name of the output descriptor: MULTCOPY

```
free file(sysprint) outdes(multcopy)
```

Example 7

Operation: Free a file and make the data set available for printing immediately after deallocation.

Known:

- The name of the file: SYSPRINT

```
free file(sysprint) spin(unalloc)
```

Example 8

Operation: Release a UNIX file.

Known:

- The ddname: OUTPUT
- The pathname: `/u/userid/file.dbp`
- The disposition: DELETE

```
free path('/u/userid/file.dbp')                    +
     pathdisp(delete)
```

HELP command

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

You cannot use the HELP command to get additional information about CLIST statements.

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

Information available through HELP

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 can use HELP again, specifying the selected command or subcommand name as an operand. You then receive:

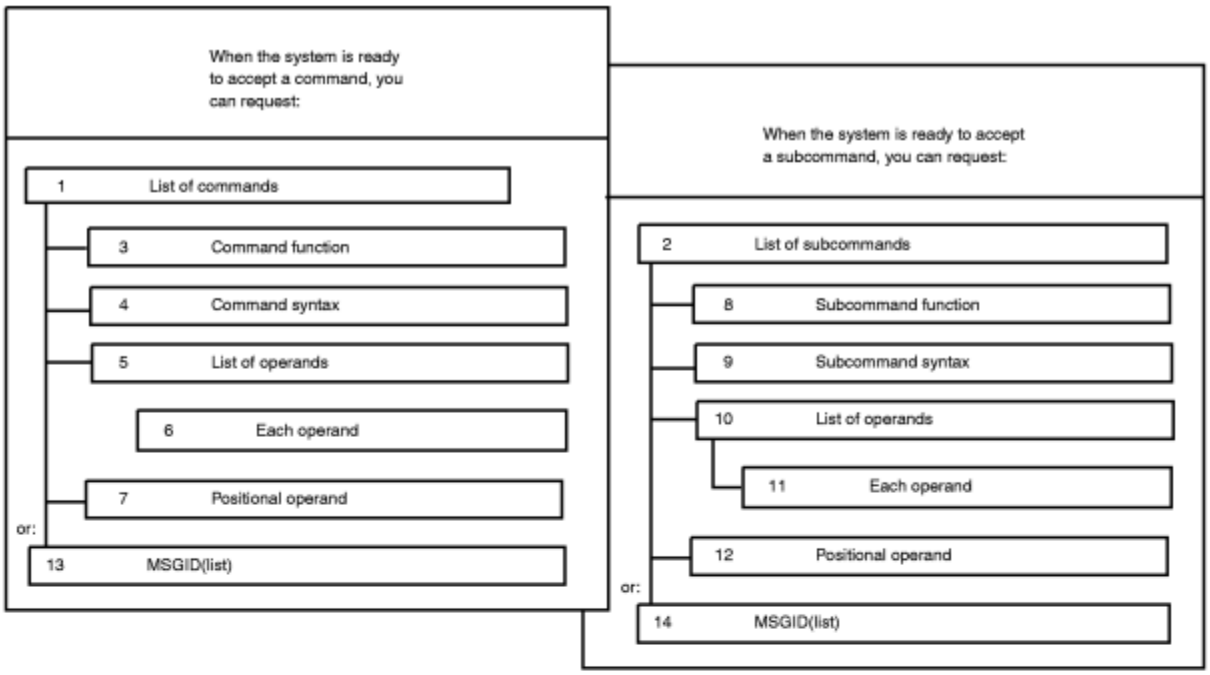
- A brief description of the function of the command or subcommand
- The format and syntax for the command or subcommand
- A description of each operand

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 can 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.

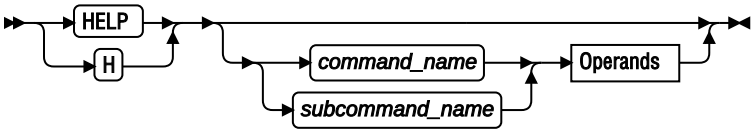
Figure 2 on page 145 shows the hierarchy of the sets of information available with the HELP command or subcommand. It also shows the form of the command or subcommand necessary to produce any particular set.



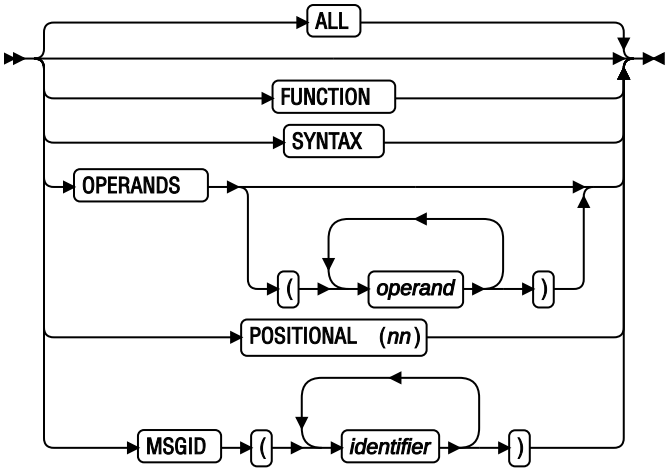
This form of the command.....	produces:	
READY mode	HELP	1
	HELP commandname	3 4 5
	HELP commandname ALL	3 4 5
	HELP commandname FUNCTION	3
	HELP commandname SYNTAX	4
	HELP commandname OPERANDS	5
	HELP commandname OPERANDS (list of keyword operands)	6
	HELP commandname POSITIONAL (positional operand number)	7
	HELP commandname MSGID (list of message IDs)	13
EDIT,OUTPUT,and TEST modes	HELP	2
	HELP subcommandname	8 9 10
	HELP subcommandname ALL	8 9 10
	HELP subcommandname FUNCTION	8
	HELP subcommandname SYNTAX	9
	HELP subcommandname OPERANDS	10
	HELP subcommandname OPERANDS (list of keyword operands)	11
	HELP subcommandname POSITIONAL (positional operand number)	12
	HELP subcommandname MSGID (list of message IDs)	14

Figure 2: Information available through the HELP command

HELP command syntax



Operands



HELP command operands

command_name* | *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 you want to know more about the syntax required to use the command or subcommand properly.

OPERANDS(*operand*)
specifies 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 are 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.

For best results, do not enter abbreviations as *operand*. HELP does not use aliases, as opposed to TSO/E commands. For example, DA is a valid alias for the DATASET operand of the ALLOCATE command, but is ambiguous if used as HELP ALLOCATE OPERANDS(DA). Therefore, specify the full operand as in HELP ALLOCATE OPERANDS(DATASET).

POSITIONAL(*nn*)
specifies that you want to obtain information about a particular positional operand of the command or subcommand. You can specify the positional operand that you want described by the number (*nn*) of the operand in the sequence of positional operands. The first positional operand needs to be identified as '1', the second as '2', and so on. You can obtain information about the positional operands of the following commands and any of their subcommands:

- ACCOUNT
- ATTRIB
- CALL
- CANCEL

- EDIT
- EXEC
- HELP
- LOGON
- MVSSERV
- OUTPUT
- RUN
- SEND
- TEST
- TRANSMIT.

ALL
specifies you want to see all information available concerning the command or subcommand. If no other keyword operand is specified, then ALL is the default.

MSGID(*list*)
specifies you want to get additional information about MVSSERV, VSBASIC, TRANSMIT, or RECEIVE messages whose message identifiers are given in the list. Information includes what caused the error and how to prevent a recurrence. You cannot specify the FUNCTION, SYNTAX, OPERANDS, or ALL operands with MSGID.

HELP command return codes

Table 19 on page 147 lists the return codes of HELP command.

Table 19: HELP command return codes

Return codes	Meaning
0	Processing successful.
12	Processing unsuccessful.

HELP command examples

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
```

Example 5

Operation: Obtain information about the ATTRIB command positional operand.

```
help attrib positional(1)
```

Example 6

Operation: Obtain information about the third positional operand of the RENUM subcommand of EDIT.

```
help renum positional(3)
```

LINK command

Use the LINK command to invoke the binder or linkage editor service programs. The binder and linkage editor convert one or more object modules (the output modules from compilers) into a load module or program object suitable for execution. In doing this, the binder and linkage editor change all symbolic addresses in the object modules into relative addresses.

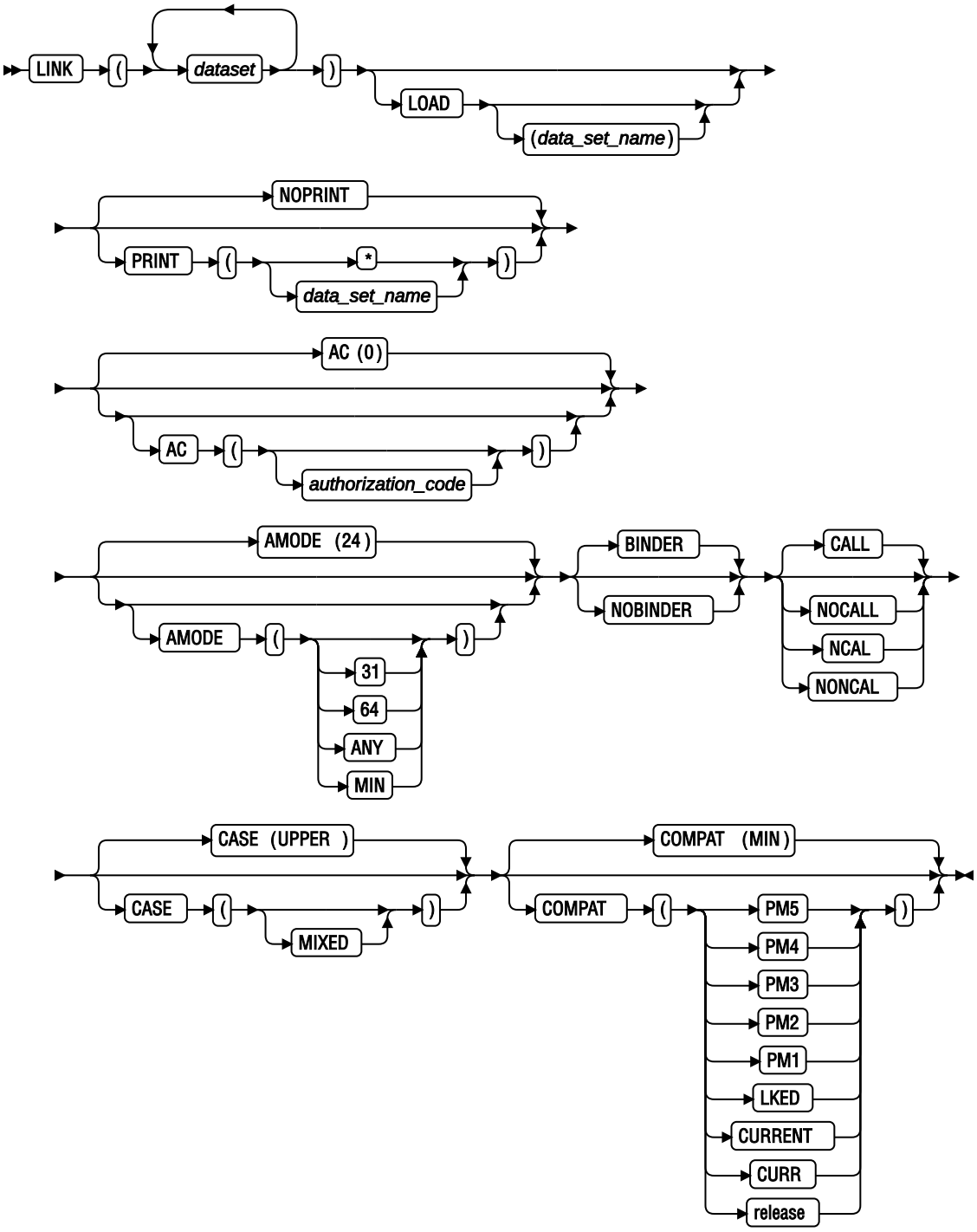
The binder and linkage editor provide a great deal of information to help you test and debug a program. This information includes a cross-reference table and a map of the module that identifies the location of control sections, entry points, and addresses. You can have this information listed at your terminal or saved in a data set.

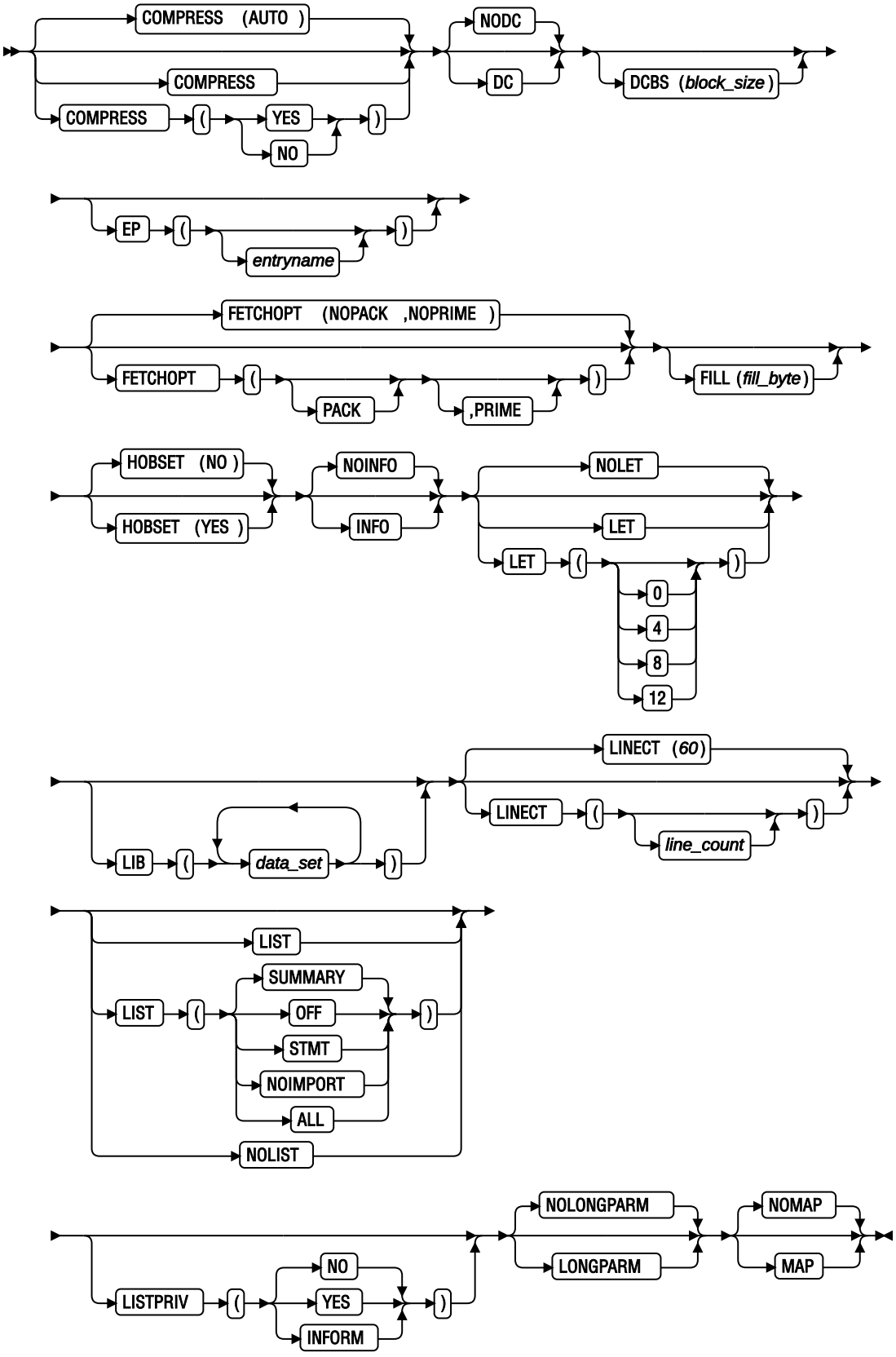
You can change binder defaults. The changes replace the defaults for the LINK command. For more information about changing binder defaults, see *z/OS MVS Program Management: User's Guide and Reference*.

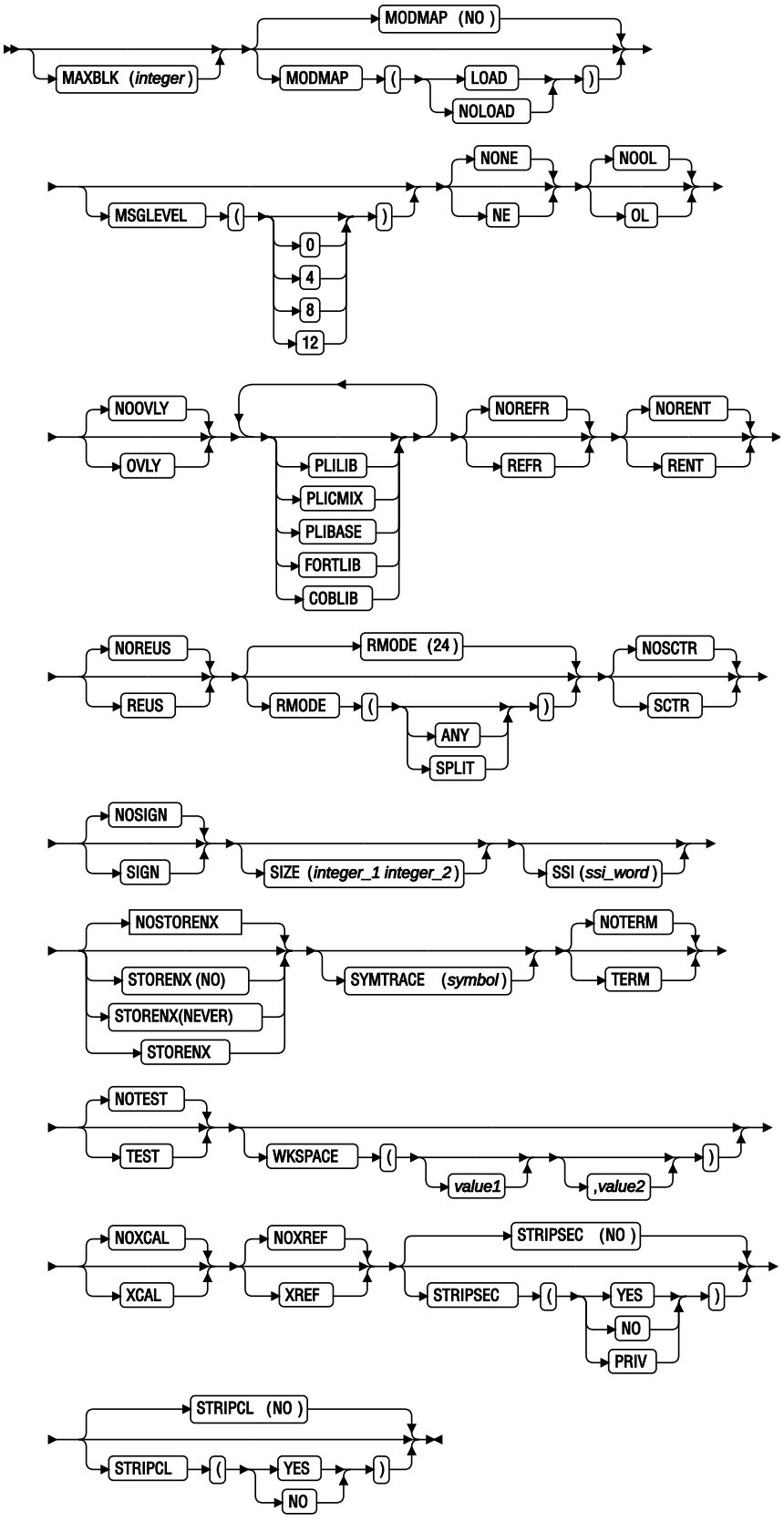
You might want to use the LOADGO command as an alternative to the LINK command, if:

- 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.
- You do not require the extensive listings produced by the binder or linkage editor.
- You do not want a load module or program object saved in a library.

LINK command syntax




COMPRESS (AUTO)
COMPRESS
COMPRESS (YES NO)
NODC
DC
DCBS (block_size)
EP (entryname)
FETCHOPT (NOPACK ,NOPRIME)
FETCHOPT (PACK ,PRIME)
FILL (fill_byte)
HOBSET (NO)
HOBSET (YES)
NOINFO
INFO
NOLET
LET
LET (0 4 8 12)
LINECT (60)
LINECT (line_count)
LIB (data_set)
LIST
LIST (SUMMARY OFF STMT NOIMPORT ALL)
NOLIST
NOLONGPARM
LONGPARM
NOMAP
MAP
LISTPRIV (NO YES INFORM)


MODMAP (NO)
MAXBLK (integer)
MODMAP
(
LOAD
NOLOAD
)
MSGLEVEL
(
0
4
8
12
)
NONE
NE
NOOL
OL
NOOVLY
OVLY
PLILIB
PLICMIX
PLIBASE
FORTLIB
COBLIB
NOREFR
REFR
NORENT
RENT
NOREUS
REUS
RMODE (24)
RMODE
(
ANY
SPLIT
)
NOSCTR
SCTR
NOSIGN
SIGN
SIZE (integer_1 integer_2)
SSI (ssi_word)
NOSTORENX
STORENX (NO)
STORENX(NEVER)
STORENX
SYMTRACE (symbol)
NOTERM
TERM
NOTEST
TEST
WKSPACE
(
value1
,value2
)
NOXCAL
XCAL
NOXREF
XREF
STRIPSEC (NO)
STRIPSEC
(
YES
NO
PRIV
)
STRIPCL (NO)
STRIPCL
(
YES
NO
)

LINK command operands

data_set
specifies the names of one or more data sets containing your object modules. The specified data sets are 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 can substitute an asterisk (*) for a data set name to indicate that you can enter control statements from your terminal. The system prompts 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 contains or will contain the load module after processing by the linkage editor. If you omit this operand, the system generates a name according to the data set naming conventions. After processing by the binder or linkage editor, the binder rejects a PDS or PDSE with a record format other than RECFM=U.

PRINT(*data_set_name*) | *NOPRINT*

PRINT(*data_set_name*)
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. If you specify LIST, MAP, or XREF operand, then PRINT is the default. If you want to have the listings displayed at your terminal, you can substitute an asterisk (*) for the data set name.

NOPRINT
specifies that no linkage editor listings are to be produced. This operand causes the MAP, XREF, and LIST options to become incorrect. If both PRINT and NOPRINT are omitted and you do not use the LIST, MAP, or XREF operand, then NOPRINT is the default.

AC(*authorization_code*)
specifies an authorization code (0-255) to maintain data security. Any non-zero value causes the program to have APF authorization if the data set is APF authorized (APF = authorized program facility).

AMODE(24 | 31 | 64 | ANY | MIN)
specifies the addressing mode for all entry points for the module to be link-edited or bound. For more information about defaulting AMODE, see *z/OS MVS Program Management: User's Guide and Reference*.

Valid AMODE values are:

24
to indicate the module is to be invoked in 24-bit addressing mode.

31
to indicate the module is to be invoked in 31-bit addressing mode.

64
to indicate the module is to be invoked in 64-bit addressing mode.

ANY
to indicates the module is to be invoked in 24-bit or 31-bit addressing mode.

MIN
causes the binder to set the AMODE to the most restrictive AMODE of all control sections in the module. In this respect, 24 is more restrictive than 31, which is more restrictive than ANY.

The MIN keyword is used only to control binder processing. It assists the binder in determining the resultant AMODE of the module. However, MIN is never used as an AMODE itself and will not

appear in the directory entry of the resultant load module or program object. MIN only has meaning when specified for PDSEs on a system with DFSMS/MVS V1R1 or later installed.

BINDER | NOBINDER

BINDER
specifies that MVS use binder services for this load module or object module rather than the linkage editor service program. The binder can be used for load modules stored in a PDS and program objects in a PDSE or Unix file. BINDER is the default.

NOBINDER
specifies that MVS not use binder services for this object module; the linkage editor service program is used to convert the object module(s) into load module(s).

CALL | NCAL | NONCAL | NOCALL

CALL | NONCAL
specifies that the automatic call mechanism is to be used to bring in additional modules for unresolved external references. CALL is the default.

NCAL | NOCALL
specifies that the automatic call mechanism is not to be used for unresolved external references.

CASE(UPPER | MIXED)

UPPER
specifies that the binder translates to uppercase all lowercase names found in input modules, control statements, and LINK parameters. UPPER is the default.

MIXED
specifies that the binder respect uppercase and lowercase names found in input modules, control statements, and LINK parameters, and treat two strings differently if a character in one string is a different case than the corresponding character in the second string. Binder keywords are always translated to uppercase.

COMPAT(MIN | PM5 | PM4 | PM3 | PM2 | PM1 | LKED | CURR | CURRENT | release)
specifies binder compatibility level.

MIN
specifies the oldest level (PM2 or higher) that supports the features in the object. This is the default.

CURRENT
indicates the latest level known to the binder.

Programs bound with this option might not be usable on older releases of z/OS.

CURR
is the abbreviation of CURRENT and has the same specification.

PM5
supports all features of lower levels plus cross-segment references by either relative or immediate instruction.

PM4
is the minimum level that can be specified if a value of 64 is specified on the AMODE option or if input modules contain 8-byte address constants or names longer than 1024 characters.

PM3
specifies that the binder create a PM3-level program object.

PM2
specifies that the binder create a PM2-level program object.

PM1
specifies that the binder create a PM1-level program object.

LKED
specifies that the binder process certain options, such as AMODE/RMODE and reusability, in a manner compatible with the linkage editor.

release

- OSV2R8 through OSV210 (same as PM3)
- ZOSV1R1 and ZOSV1R2 (same as PM3)
- ZOSV1R3 and ZOSV1R4 (same as PM4)
- ZOSV1R5 and ZOSV1R6 (adds RMODE 64 for WSA)
- ZOSV1R7 (adds compression and relative/immediate hardware instruction references across elements)
- ZOSV1R8 and ZOSV1R9 (same as PM5)
- ZOSV1R10, ZOSV1R11 and ZOSV1R12 (adds timestamp to IDRL and QY offset constant support for long displacement instruction)
- ZOSV1R13 (adds support for condition sequential RLDs)
- ZOSV2R1 (adds support for all power of 2 alignments, from 1 to 4096 bytes)

COMPRESS(YES | NO |AUTO)
Attempt to compress binder data in program object. This is only supported for ZOSV1R7 or higher format.

YES
forces format to ZOSV1R7 to allow compression.

NO
never attempts compression.

AUTO
attempts to compress only if ZOSV1R7 or higher format is required for other reasons.

DC | NODC

DC
specifies that no block in the load module is to be longer than 1024 bytes and no text block is to contain more than one control section.

NODC
specifies the DC limits do not apply. NODC is the default. This applies only to load modules stored in a PDS, not to program objects stored in a PDSE (DSNTYPE=LIBRARY) or in a Unix file. For more information see *z/OS MVS Program Management: User's Guide and Reference*

DCBS(*block_size*)
specifies the block size of the records contained in the output data set.

Note: DCBS is applicable only for load modules, not for program objects.

EP(*entryname*)
specifies the entry point of the output program. If this option is not specified and an entry control statement does not exist, the default used depends on the order in which CSECTs are bound and if a compiler specified an entry location in the END record of any object file. The first specified entry point will be used. If no entry point is specified, the entry will be the beginning of the module. The maximum value length of EP has been extended from 8 characters to 64 characters.

FETCHOPT(PACK | NOPACK, PRIME | NOPRIME)
allows control over how the module is loaded. The PACK and PRIME suboperands indicate whether the program object:

- is loaded on a double-word boundary (PACK) or on a page boundary (NOPACK)
- is (PRIME) or is not (NOPRIME) completely read into virtual storage before execution begins

Both suboperands are required for PDSEs; however, (PACK,NOPRIME) is a program object. Both suboperands apply to a program object stored in a PDSE. Only the first (PACK/NOPACK) applies to a program object stored in a Unix file. Note that PACK for a PDSE implies that the program is first page-mapped but then moved from a page boundary to a double-word boundary, forcing all data to be read. Thus the combination of (PACK,NOPRIME) is not allowed.

PACK
If PACK is specified for a PDSE, PRIME must also be specified.

A PACKed module requires a smaller amount of the user's virtual storage, but it might require more time to load. PACKed modules are aligned on doubleword boundaries.

NOPACK
specifies that the module is loaded on a page boundary. This may improve performance for a module in a PDSE when the NOPRIME option is also in effect.

PRIME
specifies that the entire program will be loaded from a PDSE into virtual storage before execution begins. This is likely to increase load time but reduce the likelihood of page faults during execution.

NOPRIME
specifies, for a program object in a PDSE, that it is only to be page-mapped before execution begins. The program code and data will be read in only when it is needed. NOPRIME is not allowed with PACK for a PDSE.

FILL*(fill_byte)*
specifies to the binder the byte value to be used to initialize storage areas in the program object. The *(fill_byte)* must be a two hexadecimal digit in the range of 0 — F.

HOBSET(NO | YES)

NO
specifies that the binder NOT set the high-order bit (HOB) in V-type adcons according to the AMODE of the target entry point. NO is the default.

YES
specifies that the binder set the high-order bit (HOB) in V-type adcons according to the AMODE of the target entry point.

INFO | NOINFO

INFO
specifies that the listing, if produced, should also include modified binder CSECT names with compile dates and most recently applied PTF for those modules that have been updated since the release was shipped.

NOINFO
specifies that the listing, if produced, should not include modified binder CSECT names with compile dates and most recently applied PTF for those modules that have been updated since the release was shipped. NOINFO is the default. This is a diagnostic aid to assist IBM personnel.

LET | LET(*sev_code*) | NOLET

LET | LET(*sev_code*)
specifies a severity code, which if exceeded, causes the module to be marked non-executable. The severity code is the aggregate error level of all calls to the binder. Valid values for severity code are 0, 4, 8, and 12. If LET is specified, it defaults to LET(8); if LET is not specified, it defaults to LET(4).

NOLET
NOLET is equivalent to LET(0).

LIB(*data_set*)
specifies one or more names of library data sets to be searched by the linkage editor or binder to locate programs referred to by the module being processed; that is, to resolve external references. When you specify more than one name, the names must be separated by a valid delimiter. If you specify more than name, the data sets are concatenated to the file name of the first data set in the list. If an input data set contains INCLUDE, LIBRARY, or AUTOCALL control statements which are intended to obtain data from the LIB data sets, the first data set in the LIB list must have been pre-allocated with the ddname in the control statements. For implicit resolution the first data set in the LIB list must have been pre-allocated with ddname SYSLIB before the LINK command. If you specify more than one name, the data sets are concatenated to the file name of the first data set and lose their individual

identity. For details on dynamic concatenation, see *z/OS MVS Programming: Authorized Assembler Services Guide*.

LINECT(60 | *line_count*)
specifies the number of lines (including heading and blank lines) contained on each page of the binder listing. The valid range is 24-200 and 0. Zero indicates a single, indefinitely long page, and values of 1-23 are forced to 24; however, there are always page ejects at the beginning of the binder listing and the start of the map, cross reference (XREF), and summary reports. LINECT defaults to 60 lines.

LIST | NOLIST | LIST(OFF | STMT | SUMMARY | NOIMPORT| ALL)
allows you to control the type of information included in the SYSPRINT data. LIST specifies a list of all linkage editor control statements is to be produced. LIST is valid for both the linkage editor and the binder. The default for LIST is SUMMARY. This is ignored if NOPRINT is specified or NOPRINT is the default. LIST, with no value, is equivalent to LIST (SUMMARY), and NOLIST is equivalent to LIST(OFF).

SUMMARY
indicates that messages, control statements and a save summary report (including processing options and module attributes) are to be printed.

OFF
specifies a listing of the linkage editor control statements is not to be produced. Only messages will be printed. In a batch environment, LIST(OFF) is equivalent to NOLIST.

STMT
indicates that messages and control statements are to be printed. In a batch environment, LIST(STMT) is equivalent to LIST.

NOIMPORT
produces the same output as SUMMARY except without IMPORT control statements.

ALL
indicates that all input activity (whether initiated by the binder service calls or control statements) and the load or save summary are to be logged.

LISTPRIV(YES | NO | INFORM)

YES
specifies that a list of unnamed (private code) sections is to be included. If the LISTPRIV operand is given without an operand, LISTPRIV(YES) is the default.

NO
specifies that a list of unnamed (private code) sections is not to be included. If LISTPRIV is not specified at all, LISTPRIV(NO) is the default.

INFORM
specifies that a list of unnamed (private code) sections is to be included.

Note: This is a diagnostic tool to detect unnamed sections, which can cause problems in rebinding. If any unnamed sections are found when running with LISTPRIV(YES), a level 8 error message will be produced. If any unnamed sections are found when running with LISTPRIV(INFORM), an informational (level 0) message will be produced.

LONGPARM | NOLONGPARM

The setting of LONGPARM has no affect on programs which are not APF authorized. Programs which are not APF authorized will be allowed to accept parameter strings longer than 100 bytes regardless if they are bound with LONGPARM or NOLONGPARM.

LONGPARM
specifies the program can accept a parameter string longer than 100 bytes. This applies to programs that are invoked using a JCL EXEC statement or UNIX System Services execmvs callable service. LONGPARM is required if the program is invoked with a parameter string of more than 100 byes and is APF authorized.

It is your responsibility to set this option only when you know that your program is capable of accepting parameters longer than 100 bytes.

NOLONGPARM
specifies the APF authorized program can not accept a parameter string longer than 100 bytes. NOLONGPARM is the default.

MAP | NOMAP

MAP
specifies 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 a map of the output module is *not* to be listed. NOMAP is the default.

MAXBLK(*integer*)
specifies the maximum text block size (in bytes) for load modules that are saved in an output library by the binder. The value range is 256-32760. If you specify a value outside this range, you receive a warning message, and the value is set to the device-dependent default value.

MAXBLK defaults to ½ of SIZE(,*integer_2*) but not less than 4096 nor more than the minimum of 32760 or the track size. This value is also compatible with that used by the linkage editor.

If neither MAXBLK nor SIZE are specified, the maximum block size defaults to the blocksize of the data set. However, if DC is also specified, the maximum block size is always set to 1024.

MODMAP(LOAD | NOLOAD | NO)
Controls whether, and where, a map of the module will be stored within the module.

LOAD
A map will be produced as CSECT IEWBMMP within the load segment which contains the module entry point.

NOLOAD
A map will be produced as section IEWBMMP in class B_MODMAP, and that class will be marked as not loadable. Binder APIs can be used to access the data.

NO
No module map will be produced. This is the default, except that if the program contains a strong reference to IEWBMMP the default will be MODMAP(LOAD).

MSGLEVEL(0 | 4 | 8 | 12)
specifies the severity level below which messages are not displayed. Valid severity levels are 0, 4, 8, and 12. If a message has a severity lower than the level indicated here, it is not printed, written to either print or terminal files, or passed to the messages exit.

NE | NONE

NE
specifies the output load module cannot be processed again by the linkage editor, loader or binder. The linkage editor does not create an external symbol dictionary. If you specify either MAP or XREF, then the NE operand is not valid for the linkage editor.

NONE
specifies the output load module can be processed again by the linkage editor, loader or binder and that an external symbol dictionary is present. NONE is the default.

OL | NOOL

OL
specifies the output load module can be brought into real storage only by the LOAD macro instruction.

NOOL
specifies the load module is not restricted to the use of the LOAD macro instruction for loading into real storage. NOOL is the default.

OVLY | NOOVLY

OVLY
specifies the output module is an overlay structure and is therefore suitable for block loading only. If you specify OVLY, do not specify SCTR. OVLY is supported for load modules and PM1–level program objects.

NOOVLY
specifies the load module is not an overlay structure. NOOVLY is the default.

PLILIB | PLIBASE | PLICMIX | FORTLIB | COBLIB

PLILIB
specifies the partitioned data set named SYS1.PL1LIB is to be searched by the LINK command to locate external symbols that are referred to by the module being processed.

PLIBASE
specifies the partitioned data set named SYS1.PLIBASE is to be searched to locate external symbols referred to by the module being processed.

PLICMIX
specifies the partitioned data set named SYS1.PLICMIX is to be searched to locate external symbols referred to by the module being processed.

FORTLIB
specifies the partitioned data set named SYS1.FORTLIB is to be searched by the LINK command to locate external symbols referred to by the module being processed.

COBLIB
specifies the partitioned data set named SYS1.COBLIB is to be searched by the LINK command to locate external symbols referred to by the module being processed.

REFR | NOREFR

REFR
specifies the load module is to be marked refreshable if the input load modules and program objects was refreshable and the OVLY operand was not specified.

NOREFR
specifies the output is not to be marked refreshable. NOREFR is the default.

RENT | NORENT

RENT
specifies the output module is marked re-enterable provided the input load modules and program objects was re-enterable and the OVLY operand was not specified.

NORENT
specifies the load module is not marked re-enterable. NORENT is the default.

REUS | NOREUS

REUS
specifies the output is to be marked serially reusable if the input load modules and program objects was re-enterable or serially reusable. The RENT and REUS operand are mutually exclusive. If the OVLY or TEST operands are specified, the REUS operand must not be specified.

NOREUS
specifies the load module is not be marked reusable. NOREUS is the default.

RMODE(MIN | 24 | ANY | 31 | SPLIT [,INITIAL | COMPAT])
specifies the residence mode for the module to be bound. If all control sections are not specified as RMODE(ANY), RMODE defaults to 24. If any section of the load module has an RMODE of 24, RMODE defaults to RMODE(24). If the RMODE operand is given without an operand, you are prompted for it. Valid RMODE values are:

MIN
to indicate that the binder is to determine the RMODE to use. Not specifying RMODE is the same as specifying RMODE(MIN,COMPAT)

24
to indicate the module must reside below the 16 MB line.

ANY
to indicate the module can reside anywhere below the 2GB bar in virtual storage.

31
RMODE(31) is a synonym for RMODE(ANY).

SPLIT
to indicate that the program object is to be split into two segments according to the RMODE of the CSECTs. SPLIT is supported only for PM2 or later format program objects.

Note: If you code SPLIT you cannot code either INITIAL or COMPAT.

INITIAL
to indicate that the RMODE option is to apply to all initial load segments.

COMPAT
to indicate that the RMODE option is to apply to only the first initial load segment.

SCTR | NOSCTR

SCTR
specifies the load module created by the linkage editor or binder can be either scatter loaded or block loaded. If you specify SCTR, do not specify OVLY. This is meaningful only for the system nucleus.

NOSCTR
specifies scatter loading is not permitted. NOSCTR is the default.

SIGN | NOSIGN

SIGN
Output program will be digitally signed by the binder, if the caller is authorized to do so.

NOSIGN
Output program will not be digitally signed.

SIZE(*integer_1*,*integer_2*)
specifies the amount of virtual storage to be used by the linkage editor. The first integer (*integer_1*) indicates the maximum allowable number of bytes. If *integer_1* is omitted, the binder does not limit its use of storage that is below the 16 MB line. *integer_2* indicates the number of bytes to be used by the linkage editor buffer as the load module buffer, which is the virtual storage area used to contain input and output data. If this operand is omitted, SIZE defaults to the size specified by your system programmer. For more information about the use of *integer_2* by the binder see the description of MAXBLK.

SSI(*ssi_word*)
specifies that the system status index (SSI) is used as a binder option. If specified, the SETSSI control statement overrides this specification. Refer to the SETSSI control statement of AMASPZAP in *z/OS MVS Diagnosis: Tools and Service Aids*, for a description of the system status index (SSI). It has to be exactly 8 hex digits.

STORENX | NOSTORENX | STORENX(YES | NO[REPLACE] | NEVER)

STORENX | STORENX(YES)
replaces the existing module of the same name in a program library with a new module, regardless of the executable status of either module. If you specify the NAME statement, you must provide the replace option (R). STORENX is supported only by the binder.

NOSTORENX | STORENX(NO[REPLACE])
is the default value, and can be specified as STORENX(NO).

STORENX(NEVER)
prevents the save of a non-executable module even when no module with the same name previously existed in the target library.

STRIPSEC(YES | NO|PRIV)

YES
Allows you to delete unreferenced sections from the module being built.

NO
Unreferenced sections will not be deleted. This is the default.

PRIV
Will delete only those unreferenced sections to which you did not assign a name.

STRIPCL(YES | NO)

YES
Unneeded classes (usually those containing debug data) will not be saved in the output module. These classes must have been marked as 'removable' in the object module and contain no adcons.

NO
All classes will be saved. This is the default.

SYMTRACE(*symbol*)
specifies a symbol that is to be resolution traced. Binder messages will inform when module references the symbol, when a library is being searched to find the symbol, where the symbol definition was eventually found, etc.

TERM | NOTERM

TERM
specifies you want error messages directed to your terminal and to the PRINT data set. TERM is the default.

Note: TERM output is a subset of PRINT output; if you specify PRINT(*) then TERM is ignored.

NOTERM
specifies you want error messages directed only to the PRINT data set and not to your terminal.

TEST | NOTEST

TEST
specifies the symbol tables created by the assembler and contained in the input modules are to be placed into the output module. This is useful only if the assembler also used the TEST option.

NOTEST
specifies no symbol table is to be retained in the output load module. NOTEST is the default.

WKSPACE(*value_1*[,*value_2*])
specifies the maximum amount of user's virtual storage available to the binder below and above 16 MB. You do not need to include this operand unless you have special virtual storage considerations such as the virtual storage between two concurrent applications needs to balance. If coded, a minimum of WKSPACE(96,1024) is suggested for all binder operations.

value_1
indicates the maximum amount (in KB) of user's virtual storage available to the binder below 16 MB in virtual storage. This value is optional; however, be certain to code a comma (,) if only *value_2* is specified.

value_2
indicates the maximum amount (in KB) of user's virtual storage available to the binder above 16 MB in virtual storage. This value is optional; however, be certain to code a comma (,) if *value_1* is not also specified.

XCAL | NOXCAL

XCAL
specifies 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 both valid and not valid exclusive calls are marked as errors. NOXCAL is the default.

XREF | NOXREF

XREF
specifies 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.

NOXREF
specifies a cross-reference listing is not to be produced. NOXREF is the default.

LINK command return codes

Table 20 on page 161 lists all the return codes of LINK command.

Table 20: LINK command return codes

Return code	Meaning
0	Processing successful.
8	Processing incomplete; system prompts you for additional information.
12	Processing unsuccessful.

LINK command examples

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 file name to be associated with the existing load module, the display at your terminal will be:

```
allocate dataset(m33payld.load) file(ld2)
link (m33thrus,*) load(m33perfo(mod2)) print(*) -
 plilib map list
IKJ76080A ENTER CONTROL STATEMENTS
 include ld2(mod1)
 entry mod2
 alias mod3
 (null line)
IKJ76111I END OF CONTROL STATEMENTS
```

Example 3

Operation: Re-specify the mode of an object module from 24-bit addressing and residence mode to 31-bit addressing and residence mode ANY.

Known:

- The name of the object module: ACCOUNT.MON.OBJ which has an addressing mode of 24-bit
- The name of the output load module: ACCOUNT.MINE.LOAD(NEWMOD)

```
link mon load(mine(newmod))amode(31) rmode(any)
```

LISTALC command

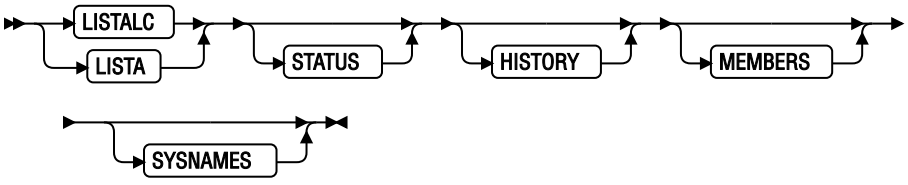
Use the LISTALC command to obtain a list of the currently allocated data sets. The LISTALC command without operands displays a list of all (partitioned and not partitioned) data set names the user has dynamically allocated and those allocated by previous TSO/E commands (issued while in the current TSO/E session). This list includes terminal data sets, indicated by the word TERMFILE, and also *attr_list_name*s created by the ATTRIB command, indicated by the word NULLFILE.

If the list should include also temporary data sets, which are created and deleted in the same job, use the SYSNAMES keyword operand to display these data sets. See *z/OS MVS JCL User's Guide* for a detailed description of permanent and temporary data sets.

If an asterisk precedes a data set name, it indicates that the data set is allocated, but marked not-in-use.

- For an allocation of a data set name with a member name (for a PS, PDS or PDSE), LISTALC prints the member name in parentheses after the data set name.
- For an allocation of a generation data set (GDS) with a relative generation number, LISTALC prints the absolute generation and version number in the data set name. It also prints the relative generation number in parentheses.

LISTALC command syntax



LISTALC command operands

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 normal termination disposition of the data set, and when listed, separated by a comma, the abnormal termination disposition. The abnormal termination disposition takes effect if an abnormal termination occurs.

  The dispositions are CATLG, DELETE, KEEP and UNCATLG. CATLG means the data set is retained and its name is in the system catalog. DELETE means 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 the data set is to be retained. UNCATLG means the data set name is removed from the catalog, but the data set is retained. For each data set managed by SMS, KEEP means CATLG and UNCATLG means DELETE.

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 whether the data set has password protection (non-VSAM only) or if the data set is RACF protected.

  Note: LISTALC HISTORY output may indicate NONE for security on a data set and LISTDS HISTORY output may indicate RACF security for the same data set. The LISTDS module is an authorized program that calls two RACF macros RACSTAT and RACHECK. LISTALC is not an authorized program and does not use the RACF macros.
- The data set organization (DSORG). The listing contains:
  - PS for sequential
  - PO for partitioned
  - IS for indexed sequential
  - DA for direct access
  - VSAM for VSAM data entries
  - DIR for any z/OS UNIX directory
  - CSPEC for any z/OS UNIX character special file
  - FILE for any z/OS UNIX regular file
  - FIFO for any z/OS UNIX FIFO special file
  - SYMLK for any z/OS UNIX symbolic link
  - TAPE for tape
  - ** for unspecified
  - ?? for any other specification

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

MEMBERS
specifies that you want to obtain the library member names of each partitioned data set having your user ID as the leftmost qualifier of the data set name. Aliases are included.

If another application is exclusively using the partitioned data set, the system displays a message and an abend code.

SYSNAMES
specifies that you want to obtain a list of all allocated data sets, including temporary data sets. For temporary data sets the system generates qualified names that start with SYS, followed by other qualifiers. See *z/OS MVS JCL User's Guide* about temporary data sets and the naming conventions applied to them.

LISTALC command return codes

Table 21 on page 164 lists the return codes of LISTALC command.

Table 21: LISTALC command return codes

Return codes	Meaning
0	Processing successful.
12	Processing unsuccessful. An error message has been issued.

LISTALC command examples

Example 1

Operation: Obtain a list of the names of the data sets allocated to you (not including the names of temporary data sets).

```
listalc
```

Example 2

Operation: Obtain a list of the names of the data sets allocated to you (not including the names of temporary data sets). 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 (including the names of temporary data sets).

```
lista members history status sysnames
```

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

```
listalc mem status sysnames history
--DSORG--CREATED-----EXPIRES-----SECURITY---DDNAME---DISP
MICHELLE.ASM
  PS            06/06/1991       00/00/0000      WRITE          DAUGHTER KEEP
NEAL.EXAMPLE
  PO            07/03/1998      00/00/0000       PROTECTEDMYSON        KEEP,KEEP
--MEMBERS--
 MEMBER1
 MEMBER2
SYS70140.T174803.RV000.TSOSPEDT.R0000001
  **     00/00/0000  00/00/0000  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 the ATTRIB command.

```
lista status
```

The output at your terminal might 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
```

Example 5

Operation: Excerpt from a job protocol showing the output from the LISTALC command with different operands, especially how LISTALC treats the temporary data sets.

```
⋮
  //JDC# JOB job card data …
  //*
  //         EXEC  PGM=IKJEFT01,DYNAMNBR=100,REGION=8M
1 //NORBERT1 DD    DSN=JDC.NORBERT,DISP=(OLD,KEEP,DELETE)
2 //NORBERT2 DD    DISP=(NEW,DELETE),SPACE=(TRK,(1,1))
3 //NORBERT3 DD    DSN=&&DSNAME,DISP=(NEW,DELETE),
  //             SPACE=(TRK,(1,1))
⋮
 IGD104I JDC.NORBERT                              RETAINED,  DDNAME=NORBERT1
 IGD105I SYS95069.T122631.RA000.JDC#.R0201039     DELETED,   DDNAME=NORBERT2
 IGD105I SYS95069.T122631.RA000.JDC#.DSNAME.H02   DELETED,   DDNAME=NORBERT3
⋮
 READY
  LISTA
 JDC.NORBERT
⋮
 READY
  LISTA STATUS
 --DDNAME---DISP--
 JDC.NORBERT
   NORBERT1 KEEP,DELETE
⋮
 READY
  LISTA SYSNAMES
 JDC.NORBERT
 SYS95069.T122631.RA000.JDC#.R0201039
 SYS95069.T122631.RA000.JDC#.DSNAME.H02
⋮
 READY
 END
```

Note the three JCL DD statements identifying:

1. A permanent data set named JDC.NORBERT
2. A temporary data set without DSN parameter at all (the system will specify a data set name)
3. A temporary data set with a name of &DSNAME.

At the bottom you see that only LISTALC with the SYSNAMES operand lists the permanent *and* temporary data sets.

LISTBC command

Use the LISTBC command to display notices and mail. Notices are messages from the operator that is intended for all users to view. Mail is messages from another user or program to a particular user. By default, notices and mail are stored in the broadcast data set, SYS1.BRODCAST. However, your installation might use user logs. If so, mail is stored in and retrieved from individual user logs. Notices are still stored in and retrieved from the broadcast data set.

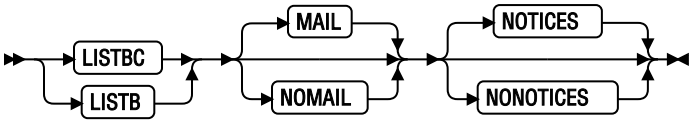
Note: When processing messages by using the SYNC, LISTBC, and SEND commands, the SYS1.BRODCAST data set is not used for 8 character user IDs. Any messages that cannot be delivered to the user's screen because the user is not logged on or the keywords on the command are only stored if user logs are enabled by using IKJTSO*xx*.

If your installation uses security labels and security checking and for SEND and LISTBC processing, MAIL messages are handled differently. When you enter the LISTBC command, LISTBC checks the security label on each MAIL message in your user log and compares it to your current security label (the security label you are logged on with). If your current security label is equal to or greater than the message's security label, the message is displayed. If your current security label is less than the message's security label, one of the following occurs:

- If you are authorized to log on with a security label that is equal to or greater than the message's security label, you receive a message stating that there is a message in your user log that you cannot view at your current security label. Log off and log on at a greater security label and issue LISTBC again.
- If you are not authorized to log on with a security label that is equal to or greater than the message's security label, the message is deleted and you do not receive notification that it was sent.

Note: For a list of the security labels you are allowed to log on with, use the RACF command SEARCH CLASS(SECLABEL).

LISTBC command syntax



LISTBC command operands

MAIL | NOMAIL

MAIL
specifies that you want to receive the messages from the broadcast data set or the user log data set that are intended specifically for you. MAIL is the default.

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

NOTICES | NONOTICES

NOTICES
specifies that you want to receive the messages from the broadcast data set that are intended for all users. NOTICES is the default.

NONOTICES
specifies that you do not want to receive the messages that are intended for all users.

LISTBC command return codes

Table 22 on page 166 lists the return codes of LISTBC command.

Table 22: LISTBC command return codes

Return codes	Meaning
0	Processing successful.
12	Processing unsuccessful.

The return codes of LISTBC command listed in Table 23 on page 167 are valid only if you have an installation-defined user log data set:

Table 23: LISTBC command return codes (installation-defined user log data set)	
Return codes	**Meaning**
0	Messages and notices are displayed.
4	Only messages are displayed.
6	One or more messages were deleted from the user log. The receiver is not authorized at a security label at which the message can be viewed.
8	Only notices are displayed.
10	User log contains messages that cannot be viewed at user's current security label.
12	No notices or messages are displayed.
16	Messages and notices are not displayed, user denied access.
20	Messages and notices are not displayed, command not authorized.
92	Messages and notices are not displayed, system error.

LISTBC command examples

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

Use the LISTCAT command 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.

In this book, "with SMS" indicates that SMS is installed and is active.

With Storage Management Subsystem, LISTCAT also lists the following Storage Management Subsystem class names:

- *Data class* contains the data set attributes related to the allocation of the data set, such as LRECL, RECFM, SPACE, and TRACKS.
- *Storage class* contains performance and availability attributes related to the storage occupied by the data set.
- *Management class* contains the data set attributes related to the migration and backup of the data set, such as performed by the Data Facility Hierarchical Storage Manager (DFSMShsm). A management class can be assigned only to a data set that also has a storage class assigned.

For information about these classes, see “SMS classes” on page 9.

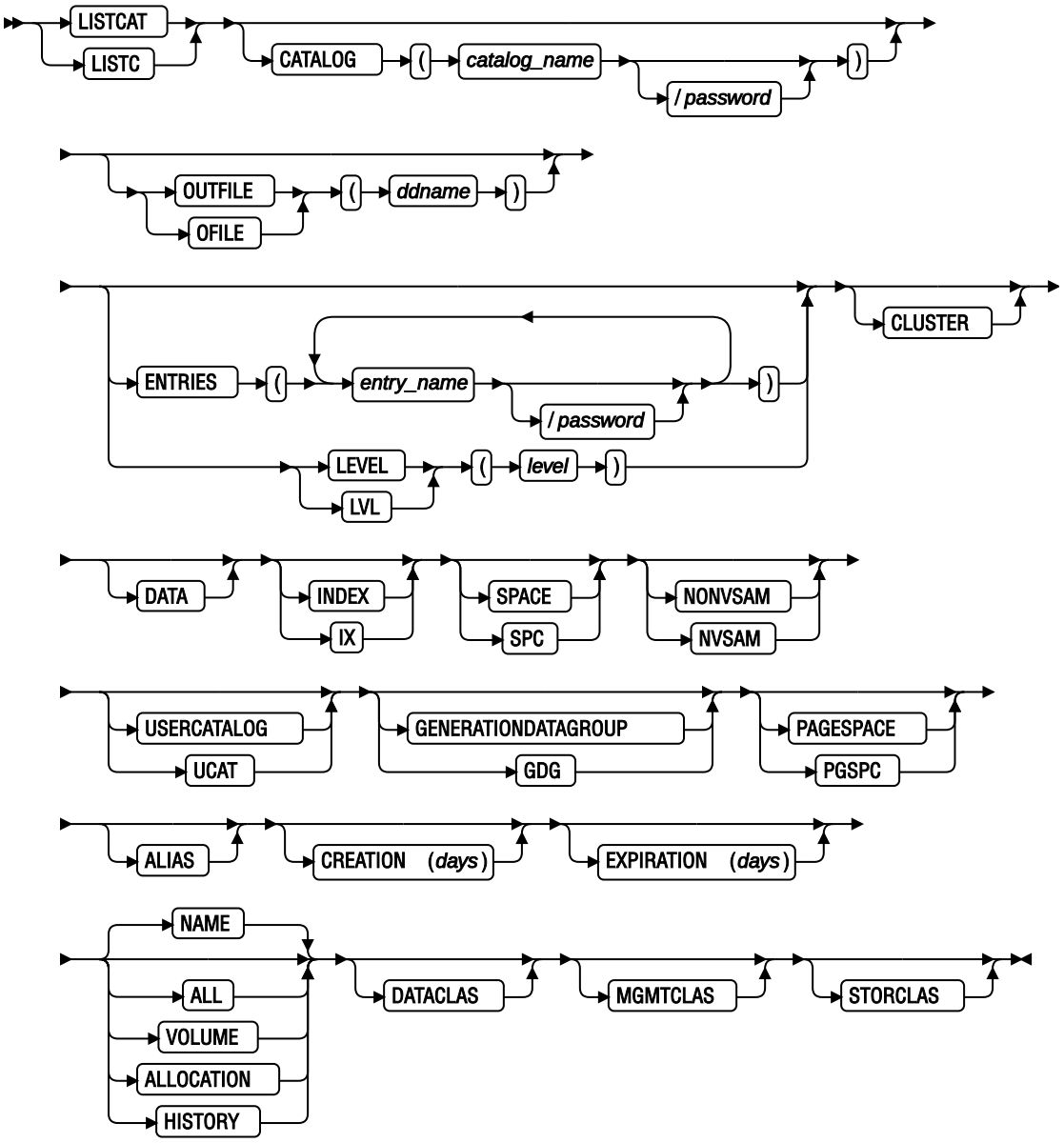
The TSO/E LISTCAT command invokes the Access Method Services command of the same name. The operand descriptions that follow provide the information required to use these services for normal TSO/E operations. The TSO/E user who wants to manipulate VSAM data sets or use the other access method

services from the terminal should see *z/OS DFSMS Access Method Services Commands*. For error message information, see the z/OS MVS System Messages books listed in Related Publications.

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

When LISTCAT is invoked and no operands are specified, the user ID or the prefix specified by the PROFILE command becomes the highest level of entry name qualification. Only those entries associated with the user ID are listed.

LISTCAT command syntax



LISTCAT command operands

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

catalog_name
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 operand or through the ENTRIES operand. If passwords are to be listed, you must specify the master password.

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

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

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

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

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

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

ENTRIES(*entry_name*/*password*)
specifies the names of the entries to be listed. If neither ENTRIES nor LEVEL is coded, only the entries associated with the user ID are listed. For more information about the ENTRIES operand, see *z/OS DFSMS Access Method Services Commands*.

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

Note: You can change a qualified name into a generic name by substituting an asterisk (*) for only 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. However, LISTCAT does not accept `*.B` as a valid generic name. The `*` is not a valid user ID for the first qualifier.

password
specifies a password when the entry to be listed is password protected and a password was not specified through the CATALOG operand. 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*) | LVL(*level*)
specifies the level of *entry_names* to be listed. If neither LEVEL nor ENTRIES is coded, only the entries associated with the user ID are listed.

CLUSTER
specifies 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 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 operand and DATA is coded, only the data component entry is listed.

INDEX | IX
specifies 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 operand and INDEX is coded, only the index component entry is listed.

SPACE | SPC
specifies entries for volumes containing data spaces defined in this VSAM catalog are to be listed. Candidate volumes are included. If entries are identified by *entry_name* or *level*, SPACE can be coded only when no other *entry_type* restriction is coded.

NONVSAM | NVSAM
specifies 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 | UCAT
specifies 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.

Note: When listing user catalogs, PROFILE NOPREFIX must be issued to ensure that *all* user catalogs will be found.

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

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

ALIAS
specifies entries for alias entries are to be listed.

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

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

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

With Storage Management Subsystem, the operands also list Storage Management Subsystem class names and the last backup data set.

ALL
specifies all fields are to be listed.

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

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

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

ALLOCATION
specifies 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.

DATACLAS
with Storage Management Subsystem, indicates that the data class of the catalog is to be listed.

MGMTCLAS
with Storage Management Subsystem, indicates that the management class of the catalog is to be listed.

STORCLAS
with Storage Management Subsystem, indicates that the storage class of the catalog is to be listed.

LISTCAT command return codes

Table 24 on page 171 lists all the return codes of LISTCAT command.

Table 24: LISTCAT command return codes

Return codes	Meaning
0	Processing successful. Informational messages might have been issued.
4	Processing successful, but a warning message has been issued.
8	Processing was completed, but specific details were bypassed.
12	Processing unsuccessful.
16	Severe error or problem encountered.

LISTDS command

Use the LISTDS command to have the attributes of specific data sets displayed at your terminal. The LISTDS command works differently, depending upon whether the data set is VSAM or non-VSAM. If you are unsure as to whether the data set is VSAM or not, enter the LISTDS command with no operands.

A VSAM data set causes the LISTDS command to display only the data set organization (DSORG), which is VSAM. Use the LISTCAT command to obtain more information about a VSAM data set.

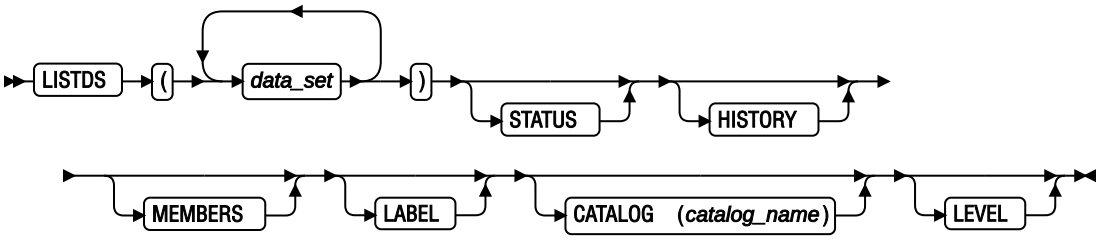
For non-VSAM data sets, you can obtain:

- The volume serial number of the DASD volume on which the data set resides.
- The logical record length (LRECL)
- The block size (BLKSIZE)
- The record format (RECFM)
- The data set organization (DSORG). The data set organization is indicated as follows:
  - PS for sequential
  - PO for partitioned
  - IS for indexed sequential
  - DA for direct access
  - VSAM for VSAM data entries
  - ** for unspecified
  - ?? for any other specification
- Directory information for members of partitioned data sets, if you specify the data set name in the form *data_set_name(member_name).*
- Creation date, expiration date, and, security attributes.
- File name and disposition

- Data set control blocks (DSCB).

Note: Data sets that are dynamically allocated by the LISTDS command are automatically freed when the command terminates, unless the data set previously was allocated with the permanent attribute. You must explicitly free dynamically allocated data sets.

LISTDS command syntax



LISTDS command operands

(*data_set*)
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 can 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 the first qualifier and `C` for the third qualifier.

Note: Do not use alias data set names with this command.

STATUS
specifies that you want the following additional information:

- The ddname currently associated with the data set.
- The normal termination disposition of the data set, and when listed, separated by a comma, the abnormal termination disposition. The abnormal termination disposition takes effect if an abnormal termination occurs.

  The keywords denoting the dispositions are CATLG, DELETE, KEEP, and UNCATLG. CATLG means the data set is cataloged. DELETE means the data set is to be removed. KEEP means the data set is to be retained. UNCATLG means the name is removed from the catalog, but the data set is retained. With a data set managed by SMS, KEEP means CATLG and UNCATLG means DELETE.

HISTORY
specifies that you want to obtain the creation and expiration dates for the specified data sets and find out whether the non-VSAM data sets are password-protected or if the data set is RACF protected.

Note: LISTALC HISTORY output may indicate NONE for security on a data set and LISTDS HISTORY output may indicate RACF security for the same data set. The LISTDS module is an authorized program that calls two RACF macros RACSTAT and RACHECK. LISTALC is not an authorized program and does not use the RACF macros.

MEMBERS
specifies that you want a list of all the members of a partitioned data set, including aliases.

LABEL
specifies that you want to have the entire data set control block (DSCB) listed at your terminal. This operand is applicable only for non-VSAM data sets on direct access devices. The list is in hexadecimal notation.

CATALOG(*catalog_name*)
specifies the user catalog that contains the names in the data set list. CATALOG is required only if the names cannot be found by normal catalog search.

LEVEL
specifies 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.

Specify only one data set list with the LEVEL option.

LISTDS command return codes

Table 25 on page 173 lists the return codes of LISTDS command.

Table 25: LISTDS command return codes

Return codes	Meaning
0	Processing successful.
12	Processing unsuccessful. An error message has been issued.

LISTDS command examples

Example 1

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

Known:

- The data set name: ZALD58.CIR.OBJ

```
listds cir
```

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

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

LOADGO command

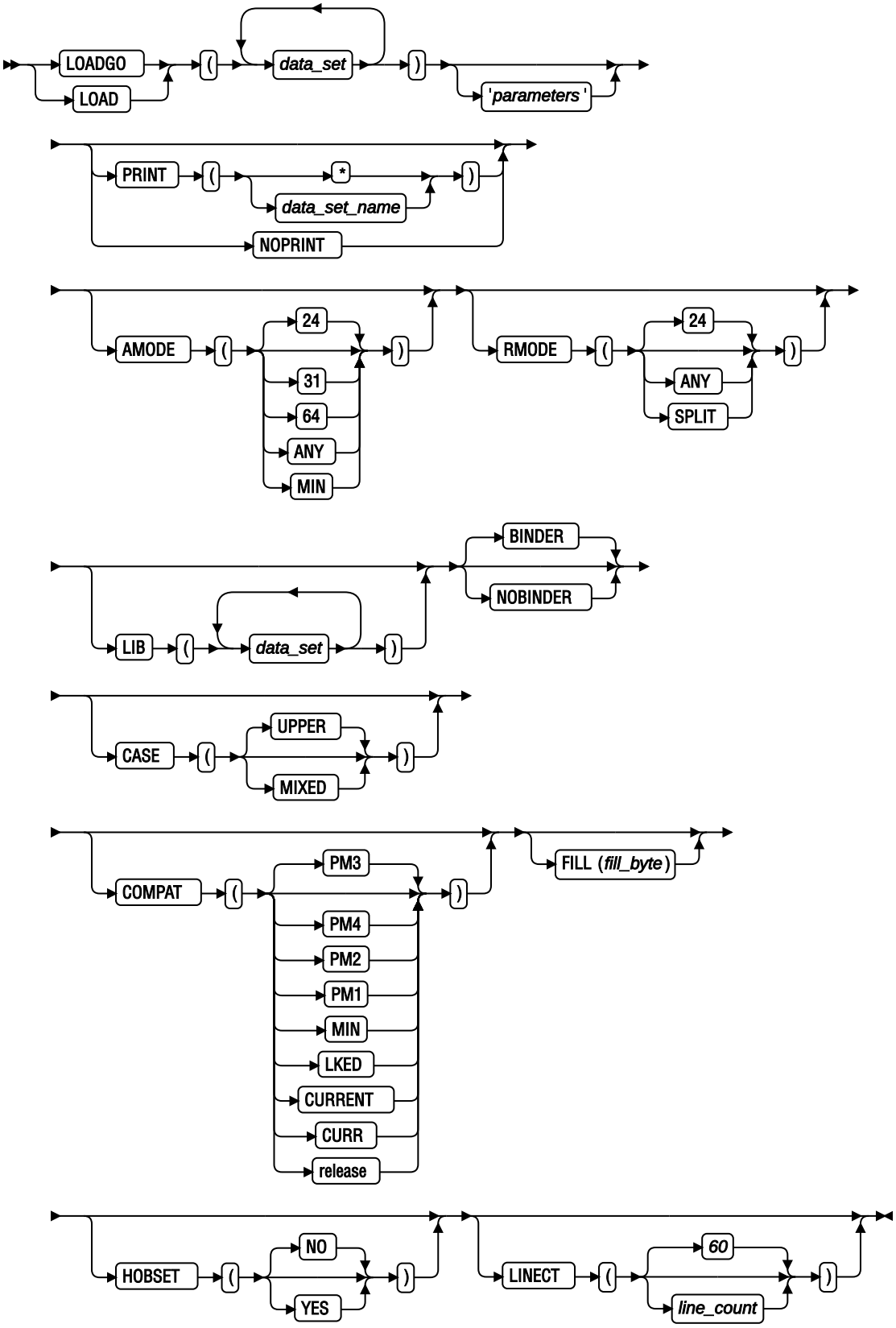
Use the LOADGO command to load a compiled or assembled program into virtual storage and begin execution.

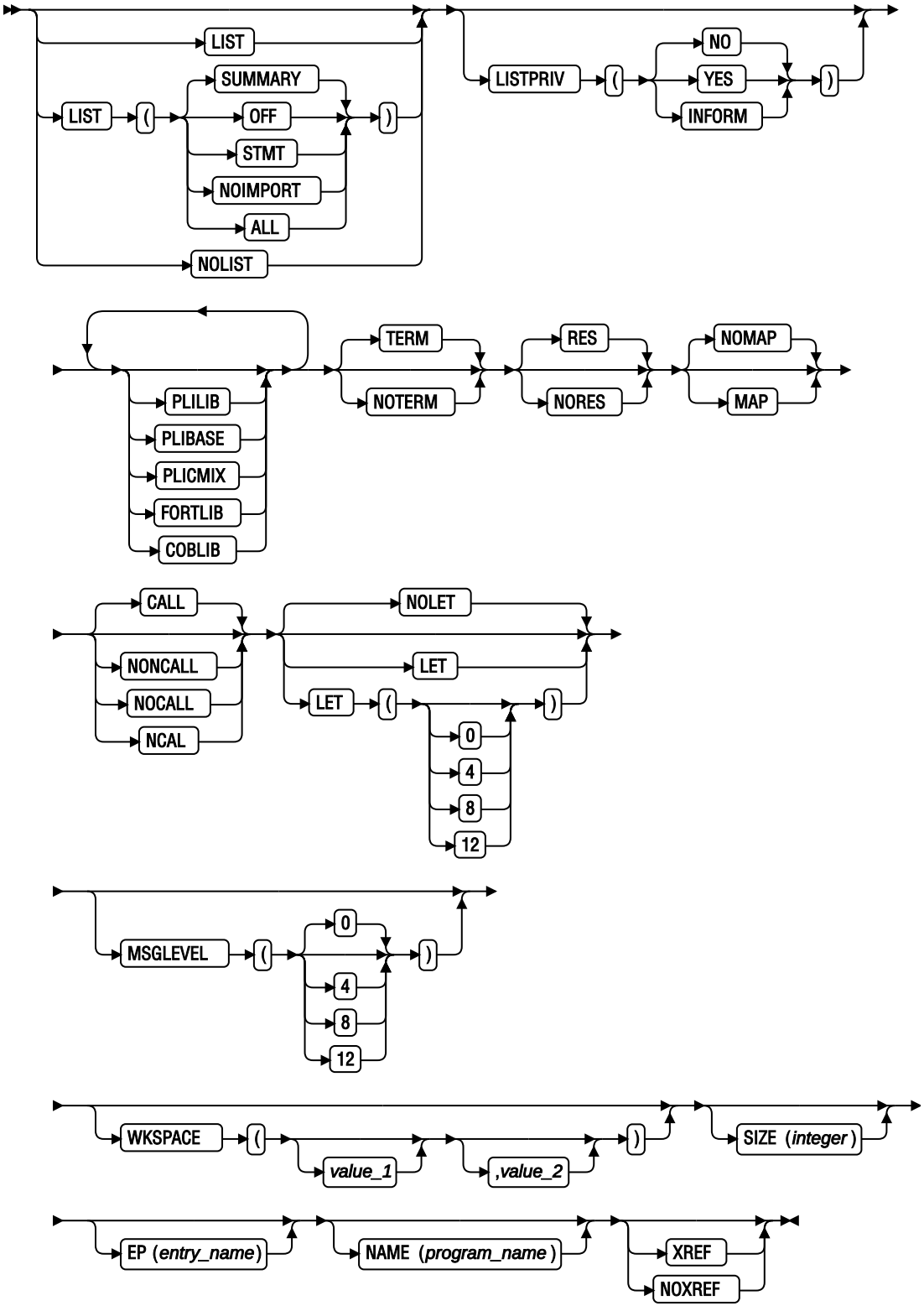
The LOADGO command loads object modules produced by a compiler or assembler, and load modules or program objects produced by the linkage editor or binder. If you want to load and execute a single load module, the CALL command is more efficient. The LOADGO command also searches a call library (SYSLIB) or a resident link pack area, or both, to resolve external references.

The LOADGO command invokes the binder or the batch loader to accomplish this function, and combines basic editing and loading services of the binder or 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 or program objects for program libraries. If NOBINDER is specified, LOADGO does not process linkage editor control statements such as INCLUDE, NAME, OVERLAY, and so on.

LOADGO command syntax





LOADGO command operands

(*data_set*)

specifies the names of one or more object modules, load modules and /or program objects to be loaded and executed. The names can 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* | *) | NOPRINT

PRINT(*data_set_name* | *)
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 is named according to the data set naming conventions. You can substitute an asterisk (*) for the data set name, if you want to have the listings displayed at your terminal. If you specify the MAP operand, then PRINT is the default.

NOPRINT
specifies no listings are to be produced. This operand suppresses the MAP operand. If both PRINT and NOPRINT are omitted and you do not use the MAP operand, then NOPRINT is the default.

AMODE(24 | 31 | 64 | ANY | MIN)
specifies the addressing mode for the module to be loaded. If the AMODE operand is not specified, AMODE defaults to the AMODE of the main entry point. Valid AMODE values are:

24
indicates that the module is invoked in 24-bit addressing mode.

31
indicates that the module is invoked in 31-bit addressing mode.

64
indicates that the module is invoked in 64-bit addressing mode.

ANY
indicates that the module is invoked in either 24-bit or 31-bit addressing mode.

MIN
causes the binder to set the AMODE to the most restrictive AMODE of all control sections in the module. In this respect, 24 is more restrictive than 31, which is more restrictive than ANY.

The MIN keyword is used only to control binder processing. It assists the binder in determining the resultant AMODE of the module. However, MIN is never used as an AMODE itself and will not appear in the directory entry of the resultant load module or program object.

For more information, see *z/OS MVS Program Management: User's Guide and Reference*.

RMODE(24 | ANY | SPLIT)
specifies the residence mode for the module to be loaded. If all control sections are not specified as RMODE(ANY), RMODE defaults to 24. If any section of the load module has an RMODE(24), RMODE defaults to 24. Valid RMODE values are:

24
to indicate the module must reside below the 16 MB line

ANY
to indicate the module can reside anywhere below the 2GB bar in virtual storage

SPLIT
to indicate that the program object is to be split into two segments according to the RMODE of the CSECTs. SPLIT is supported only for PM2 or PM3 format program objects.

LIB(*data_set*)
specifies 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).

BINDER | NOBINDER

BINDER
specifies that MVS use binder services for this load module or object module rather than the loader service program. The binder can be used for to read modules stored in a PDS and program objects in a PDSE or Unix file. BINDER is the default.

NOBINDER
specifies that MVS not use binder services for this object module; the loader service program is used to process the object module(s) into load module(s).

CASE(UPPER | MIXED)

UPPER
specifies that the binder translates to uppercase all lowercase names found in input modules, binder control statements, and LOADGO parameters. UPPER is the default.

MIXED
specifies that the binder respect uppercase and lowercase names found in input modules, control statements, call parameters, and LOADGO parameters. If MIXED is specified, the binder treats two strings differently if any character in one string is a different case than the corresponding character in the second string. binder keywords are always translated to uppercase.

COMPAT(MIN | CURR | CURRENT | LKED | PM1 | PM2 | PM3 | PM4 | release)
specifies binder compatibility level.

MIN
indicates the oldest level (PM2 or higher) that supports the features in the object. This is the default.

CURRENT
indicates the latest level know to the binder.

CURR
is the abbreviation of CURRENT and has the same specification.

LKED
specifies that the binder process certain options, such as AMODE/RMODE and reusability, in a manner compatible with the linkage editor.

PM1
specifies that the binder create a PM1-level program object or load module.

PM2
specifies that the binder create a PM2-level program object.

PM3
specifies that the binder create a PM3-level program object. PM3 is the default.

PM4
is the minimum level that can be specified. A value of 64 is specified on the AMODE option. Input modules contain 8-byte address constants and names longer than 1024 characters.

release

- OSV2R8 through OSV210 (same as PM3)
- ZOSV1R1 and ZOSV1R2 (same as PM3)
- ZOSV1R3 and ZOSV1R4 (same as PM4)
- ZOSV1R5 and ZOSV1R6 (adds RMODE 64 for WSA)
- ZOSV1R7 (adds compression and relative/immediate hardware instruction references across elements)

FILL*(fill_byte)*
specifies to the binder the byte value to be used to initialize uninitilalized storage areas in the loaded program. This must be two hexadecimal digits or the word NONE.

HOBSET(NO | YES)

NO
specifies that the binder NOT set the high-order bit (HOB) in V-type adcons according to the AMODE of the target entry point. NO is the default.

YES
specifies that the binder set the high-order bit (HOB) in V-type adcons according to the AMODE of the target entry point.

LINECT
specifies the number of lines (including heading and blank lines) contained on each page of the binder listing. The valid range is 24-200 and 0. Zero indicates a single, indefinitely long page, and values of 1-23 are forced to 24; however, there are always page ejects at the beginning of the binder listing and the start of the map, cross reference (XREF), and summary reports. LINECT defaults to 60 lines.

LIST | NOLIST | LIST(OFF | STMT | SUMMARY | NOIMPORT| ALL)
allows you to control the type of information included in the SYSPRINT data. LIST is valid for both the linkage editor and the binder. LIST, with no value, is equivalent to LIST (SUMMARY), and NOLIST is equivalent to LIST(OFF).

OFF
indicates that only messages will be printed. In a batch environment, LIST(OFF) is equivalent to NOLIST.

STMT
indicates that messages and control statements to be printed. In a batch environment, LIST(STMT) is equivalent to LIST.

NOIMPORT
produces the same output as SUMMARY except without IMPORT control statements.

SUMMARY
indicates that messages, control statements, and a load summary report (including processing options and module attributes) are to be printed.

ALL
indicates that all input activity (whether initiated by binder service calls or control statements) and the load or save summary be logged.

PLILIB
specifies the partitioned data set named SYS1.PL1LIB is to be searched to locate external symbols referred to by the module being processed.

PLIBASE
specifies the partitioned data set named SYS1.PLIBASE is to be searched to locate external symbols referred to by the module being processed.

PLICMIX
specifies the partitioned data set named SYS1.PLICMIX is to be searched to locate external symbols referred to by the module being processed.

FORTLIB
specifies the partitioned data set named SYS1.FORTLIB is to be searched to locate external symbols referred to by the module being processed.

COBLIB
specifies the partitioned data set named SYS1.COBLIB is to be searched to locate external symbols referred to by the module being processed.

TERM | NOTERM

TERM
specifies that you want any error messages directed to your terminal and the PRINT data set. If both TERM and NOTERM are omitted, then TERM is the default.

NOTERM
specifies that you want any error messages directed only to the PRINT data set.

RES | NORES

RES
specifies the link pack area is to be searched for load modules (referred to by the module being processed) before the specified libraries are searched. If both RES and NORES are omitted, then RES is the default. If you specify the NOCALL operand, the RES operand is not valid.

NORES
specifies the link pack area is not to be searched to locate modules referred to by the module being processed.

MAP | NOMAP

MAP
specifies 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 external names and addresses are not to be contained in the PRINT data set. If both MAP and NOMAP are omitted, then NOMAP is the default.

CALL | NONCAL | NOCALL | NCAL

CALL | NONCAL
specifies the data set specified in the LIB operand is to be searched to locate load modules referred to by the module being processed. CALL is the default.

NOCALL | NCAL
specifies the data set specified by the LIB operand is not to be searched to locate modules that are referred to by the module being processed. The RES operand is not valid when you specify NOCALL.

LET(*sev_code*)
specifies a severity code, which if exceeded, will prevent execution of the module. The severity code is the aggregate error level of all calls to the binder. Valid values for severity code are 0, 4, 8, and 12. If LET is specified, it defaults to LET(8); if LET is not specified it defaults to LET(4). NOLET is equivalent to LET(0).

MSGLEVEL
specifies the severity level below which messages are not displayed. Valid severity levels are 0, 4, 8, and 12. If a message has a severity lower than the level indicated here, it is not printed, written to either print or terminal files, or passed to the messages exit.

WKSPACE(*value_1*[,*value_2*])
specifies the maximum amount of user's virtual storage available to the binder below and above 16 MB. You do not need to include this operand unless you have special virtual storage considerations such as the virtual storage between two concurrent applications needs to balance. If coded, a minimum of WKSPACE(96,1024) is suggested for all binder operations.

value_1
indicates the maximum amount (in KB) of user's virtual storage available to the binder below 16 MB in virtual storage. This value is optional; however, be certain to code a comma (,) if only *value_2* is specified.

value_2
indicates the maximum amount (in KB) of user's virtual storage available to the binder above 16 MB in virtual storage. This value is optional; however, be certain to code a comma (,) if *value_1* is not also specified.

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 by your system programmer.

EP(*entry_name*)
specifies the external name for the entry point to the loaded program. If the entry point of the loaded program is in a load module, you must specify the EP operand. The maximum value length of EP has been extended from 8 characters to 64 characters.

NAME(*program_name*)
specifies the name that you want assigned to the loaded program.

XREF | NOXREF

XREF
A cross-reference list of external symbol usage is written to the print destination. A map of symbol locations must be requested separately if desired.

NOXREF
No cross-reference of external symbol usage is produced.

LISTPRIV(YES | NO | INFORM)

NO
specifies that a list of unnamed (private code) sections is not to be included. If LISTPRIV is not specified at all, LISTPRIV(NO) is the default.

YES
specifies that a list of unnamed (private code) sections is to be included. If the LISTPRIV operand is given without an operand, LISTPRIV(YES) is the default.

INFORM
specifies that a list of unnamed (private code) sections is to be included.

Note: This is a diagnostic tool to detect unnamed sections, which can cause problems in rebinding. If any unnamed sections are found when running with LISTPRIV(YES), a level 8 error message will be produced. If any unnamed sections are found when running with LISTPRIV(INFORM), an informational (level 0) message will be produced.

LOADGO command return codes

Table 26 on page 180 lists the return codes of LOADGO command.

Table 26: LOADGO command return codes

Return codes	Meaning
0	Processing successful.
8	Processing incomplete, system prompts you for additional information.
12	Processing unsuccessful.

LOADGO command examples

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)
```

Example 3

Operation: Combine and execute several object and load modules with differing AMODE and RMODE attributes. The new load module should execute in 31-bit addressing mode and be loaded anywhere in storage.

Known:

- The name of the main routine, a load module in 24-bit addressing mode: MY.PROG.LOAD(MAIN)
- The names of two subroutines, which are updated with changes before loading; both are AMODE(31) and RMODE(ANY): MY.SUB1.OBJ, MY.SUB2.OBJ

```
load (sub1 sub2 'my.prog.load(main)') print (*) amode(31)
rmode(any)
```

LOGOFF command

Use the LOGOFF command to terminate your terminal session. When you enter the LOGOFF command, the system frees all the data sets allocated to you. Data remaining in storage is lost.

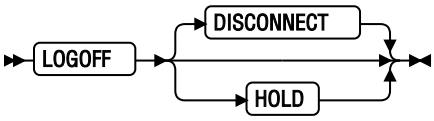
If you intend to enter the LOGON command immediately to begin a new session using different attributes, you are not required to LOGOFF. Instead, you can just enter the LOGON command as you need to enter any other command.

If your terminal is a Systems Network Architecture (SNA) terminal that uses VTAM®, you might be required to use a format different from the one described here. Your system programmer should provide you with this information.

When the LOGOFF command is executed in the background, your TSO/E session is terminated normally. Any remaining commands in the input stream are ignored.

The TSO segment, which includes the TSO profile, will only be written to the RACF database if it has changed during the TSO/E session where the user is issuing LOGOFF.

LOGOFF command syntax



LOGOFF command operands

DISCONNECT
specifies the line is to be disconnected following logoff. If no operand is specified, then DISCONNECT is the default.

HOLD [3]
specifies the line is not to be disconnected following logoff.

LOGOFF command return codes

Table 27 on page 182 lists the return codes of the LOGOFF command.

[3] Not supported with terminals that use VTAM.

Table 27: LOGOFF command return codes	
Return code	**Meaning**
0	Processing successful.

LOGOFF command examples

Example 1

Operation: Terminate your terminal session.

```
logoff
```

LOGON command

Use the LOGON command to start a terminal session. If you are not familiar with the logon process, see *z/OS TSO/E User's Guide*.

Before you can use the LOGON command, your installation must provide you with certain basic information:

- Your user identification (1 to 8 characters).
- A password (1-8 characters, unless password phrase support is active). For a RACF-defined user, your installation assigns a RACF password for you.

  Note: TSO/E does not allow you to log on with password phrases that contain leading or trailing blanks.
- An account number, if required by your installation.
- A procedure name, if required by your installation.
- For a RACF-defined user, a GROUP name, if required by your installation.

The information that you enter with the LOGON command and its operands is used by the system to start and control your time sharing session. At least, you are required to identify yourself to the system with the *user_identity* operand. Mostly, you are required to enter a password. Other operands are optional, or provide default values, and allow you to control the way that your session is to work. For example, you can specify whether you want to receive messages from the system or from other users while your session is active.

Full-Screen LOGON versus line mode LOGON

There are two types of LOGON command processing: full-screen LOGON command processing and line mode LOGON command processing.

- If you are an IBM 3270 terminal user, using a display format of 24 x 80 (24 lines of data by 80 characters on a line) or larger, you must use full-screen logon. Full-screen logon users need only enter `logon user_id`. TSO/E then displays a full-screen logon menu with appropriate entry fields for both RACF and non-RACF defined users.

  If you enter more parameters than *user_id* on the LOGON command, TSO/E accepts and processes them with the exception of the new password field. TSO/E requires the password entries to be entered on the logon menu for full-screen logon processing.
- If your terminal is such that full-screen LOGON command processing cannot be used, then all of the logon information must be specified in line mode and you might be prompted by the system to enter values for certain operands that are required by your installation.

Full-Screen LOGON processing

After you have issued a LOGON command the full-screen logon command processing performs the following:

- If your installation has PasswordPreprompt active, a line mode prompt for password is displayed. If either user ID or password are incorrect, the system responds with message `user ID or password not authorized` and terminates.
- It displays a menu with the previous session's logon parameter values. Logon command parameters entered on the LOGON command override any default values from the previous session.
- If not previously entered, it requests that you enter a password. If you enter an invalid password, the system will prompt you to re-enter it after you have pressed the Enter key.
- If your user ID is defined to RACF, it allows you to enter a *new password* on the logon panel.

  If you have entered a new password, and after pressing the Enter key, logon processing prompts you to re-enter the new password in the same field a second time to verify the password. If both entries of the new password match, logon processing proceeds. Otherwise, logon processing displays a message that the password verification failed. If this occurs, do one of the following:

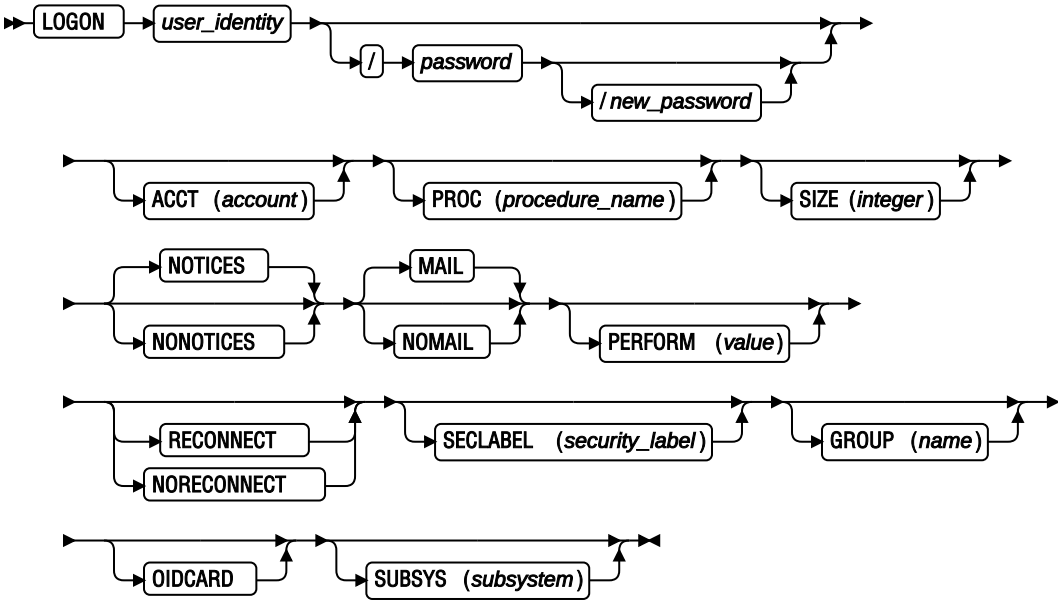
  – If you want to change your password, or if the system requires you to change it because the old password is expired, enter again a new password. The system will prompt you to enter the new password again for verification.
  – If you do not want to change your password, press Enter without entering a new password.

  Note: Full-screen logon processing allows you to enter parameter values in any of the menu fields before pressing the Enter key. Actual field verification takes place after the Enter key is pressed. If you have entered a not valid password, the system responds with message `Password not authorized for user ID` after you press Enter, awaiting the correct password. Should you have entered a not valid password *and* a new password, the system responds with message `Password not authorized for user ID. New password ignored` after you press Enter, and ignores any entry you did in the new password field. You need to re-enter the new password after you have corrected the original "Password not authorized ..." problem.
- Further, full-screen logon allows for modification and entry of logon parameter values. You can type over existing values on the menu displayed. Existing values are either from a previous session logon or from the current LOGON command parameters.
- It displays RACF entry fields, if RACF is installed and active and the user ID is RACF-defined.
- Full-screen logon allows you to enter a single TSO/E command up to 80 characters long on the logon menu. This command is executed *after* any command entered in the PARM field on the EXEC statement of the LOGON procedure. This command is also remembered from session to session.
- It displays help information for all logon parameters whenever you can enter USERID, PASSWORD, or RACF password. Help information is displayed for the entry being prompted for and in all cases, except for the PASSWORD entry fields, displays the user entered data as well.

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

When the LOGON command is executed in the background, the system ignores any remaining commands in the input stream and it has no effect on your foreground TSO/E session, if you have one.

LOGON command syntax



LOGON command operands

user_identity*/*password*/*new_password
Specifies your user identification and, if required, a valid password or new password. Your user identification must be separated from the password by a slash (/) and, optionally, one or more standard delimiters (tab, blank, or comma). Your identification and password must match the identification contained in the system's user attribute data set (UADS), if you are not RACF-defined. If you are RACF-defined, you must enter the password defined in the RACF data set as the value for password. The new password specifies the password that is to replace the current password. *new_password* must be separated from *password* by a slash(/) and, optionally, one or more standard delimiters (tab, blank, or comma). *new_password* is 1 to 8 alphanumeric characters long. This operand is ignored for non-RACF defined users. (Printing is suppressed for some types of terminals when you respond to a prompt for a password.)

With z/OS V1R7 or later, the *password* and *new_password* can be in mixed case, if your installation has enabled RACF mixed case password support.

ACCT(*account*)
Specifies the account number that is 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 prompts you for it.

For TSO/E, 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. If you omit this operand, the UADS contains a default value for your region size. The UADS also contains a value for the maximum region size that you are allowed. If you specify a region size exceeding the maximum region size allowed by the UADS (in this case, the UADS value MAXSIZE is used), then this operand is rejected.

NOTICES | NONOTICES
Specifies whether messages intended for all terminal users are to be listed at your terminal during logon processing.

NOTICES
Specifies messages are to be listed. NOTICES is the default.

NONOTICES
Specifies no messages are to be listed.

MAIL | NOMAIL
Specifies whether you want messages intended specifically for you to be displayed at your terminal during logon processing.

MAIL
Specifies that you want messages to be displayed. MAIL is the default.

NOMAIL
Specifies that you do not want messages to be displayed.

PERFORM(*value*)
Specifies the performance group to be used for the terminal session. The value must be an integer from 1-999. However, the line mode LOGON limit is 255. The default value is determined by the individual installation.

RECONNECT | NORECONNECT

RECONNECT
By default in z/OS V1R11, the LOGONHERE support for the LOGON command is on; you can reconnect to your session even if no disconnection has been detected. Using the LOGONHERE support, you can easily switch from one workstation to another or re-establish your session with a new IP address. If LOGONHERE support is not active, you can only reconnect to your session if it has been previously disconnected. For more information about activating or deactivating LOGONHERE support, see *z/OS TSO/E Customization*. If you are RACF-defined, RECONNECT remains in effect across sessions until you specify NORECONNECT. However, if the UADS contains your user information, then RECONNECT does not remain in effect across sessions. If you have specified a password in the disconnected session, you must specify the same password with the RECONNECT option. If RECONNECT is specified, then any operands other than user ID and password are ignored.

NORECONNECT
Specifies that you do not want automatic reconnect to be in effect for the session you are logging on to.

SECLABEL(*security_label*)
Specifies a security label for your TSO/E session. The SECLABEL (security label) may be 1 to 8 alphanumeric characters. The first character must be alphabetic or one of the special characters #, $, or @. SECLABEL is recognized only if your installation is using security labels and security checking and you are RACF-defined. If you specify a SECLABEL for which you are not authorized, you receive an error message and are prompted for another SECLABEL. If you do not specify SECLABEL on the LOGON command, RACF uses the default set by your administrator.

If you log on to TSO/E in line mode and you want to use a SECLABEL other than the default, you must include it each time you log on.

GROUP(*name*)
Specifies a 1 to 8 character ID composed of alphanumeric characters. The first character must be alphabetic or one of the special characters #, $, or @. This operand is valid only for RACF-defined users. It will be ignored for users not defined to RACF.

OIDCARD
Specifies the operator identification card is to be prompted for after the LOGON command has been entered. This operand is valid only for RACF-defined users.

If you are not defined to RACF and enter this keyword, you are prompted for an operator identification card. However, any data you enter is ignored. You can also enter a null line in response to the prompt.

SUBSYS(*subsystem*)
Specifies the subsystem where this LOGON is directed.

LOGON command return codes

Table 28 on page 186 lists the return codes of the LOGON command.

Table 28: LOGON command return codes

Return code	Meaning
0	Processing successful.

LOGON command examples

Example 1

Operation: Start 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: Start a terminal session.

Known:

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

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

Example 3

Operation: Start a terminal session.

Known:

- Your user identification and password: WRRID/XTD18
- Your account number: 347971
- The name of a cataloged procedure: RS832
- Your virtual storage space requirement: 90K bytes
- The security label for the session: CONFDNTL

```
logon wrrid/xtd18 acct(347971) proc(rs832)-
 size(90) seclabel(confdntl)
```

MVSSERV command

Use the MVSSERV command to start a session between an IBM Personal Computer (PC) and an IBM host computer running TSO/E on MVS. The is a set of programs that allows a PC user to request services from a host program. The PC requesting program is referred to as the *requester*. The host program that executes the corresponding service is referred to as the *server*. For more information about IBM-supplied servers that you can use, see *Introduction to IBM System/370 to IBM Personal Computer Enhanced Connectivity Facilities*.

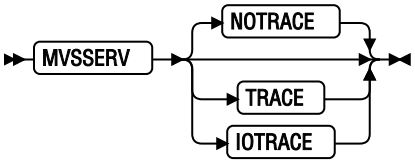
By using the operands on the MVSSERV command, you can accumulate all, some, or no diagnostic information in a *trace data set*. The diagnostic information includes the following:

- Informational and error messages
- An execution path table that tracks module calls
- Requests and replies sent between the PC and the host

However, before using MVSSERV, you must have certain pre-allocated data sets. Your installation may have already pre-allocated those data sets for you. They are described in *z/OS TSO/E Guide to SRPI*. The guide also describes how to write and install servers.

Note: MVSSERV is not supported under z/OSMF ISPF.

MVSSERV command syntax



MVSSERV command operands

NOTRACE
runs MVSSERV without sending messages to a trace data set. Use NOTRACE for production work. When testing or debugging a program, use TRACE or, preferably, IOTRACE, to obtain complete diagnostic information about the MVSSERV session. NOTRACE is the default.

TRACE
records all terminal messages and most diagnostic messages in a trace data set. The TRACE operand requires a pre-allocated trace data set in which to store the messages. Your system programmer may have allocated the data set for you. See *z/OS TSO/E Guide to SRPI* for information about how to allocate the trace data set.

IOTRACE
records all terminal messages and all diagnostic messages in a trace data set. In addition to the messages recorded with the TRACE operand, the IOTRACE operand records communication information about data flow and data sent between the host and the PC.

MVSSERV command return codes

Table 29 on page 187 lists the return codes of MVSSERV command.

Table 29: MVSSERV command return codes

Return codes	Meaning
0	Processing successful.

Table 29: MVSSERV command return codes (continued)	
Return codes	**Meaning**
4	Processing unsuccessful.

MVSSERV command examples

Example 1

Operation: Start a session program for production.

```
MVSSERV
```

or

```
MVSSERV NOTRACE
```

Example 2

Operation: Start a session and record all terminal messages and all diagnostic messages in the trace data set.

```
MVSSERV IOTRACE
```

Example 3

Operation: Start a session and record all terminal messages and some diagnostic messages in the trace data set.

OUTDES command

Use the OUTDES command to create or reuse a dynamic output descriptor. An output descriptor defines output characteristics that will be referenced by a SYSOUT data set. OUTPUT JCL statements in the LOGON procedure can also be used to define output descriptors.

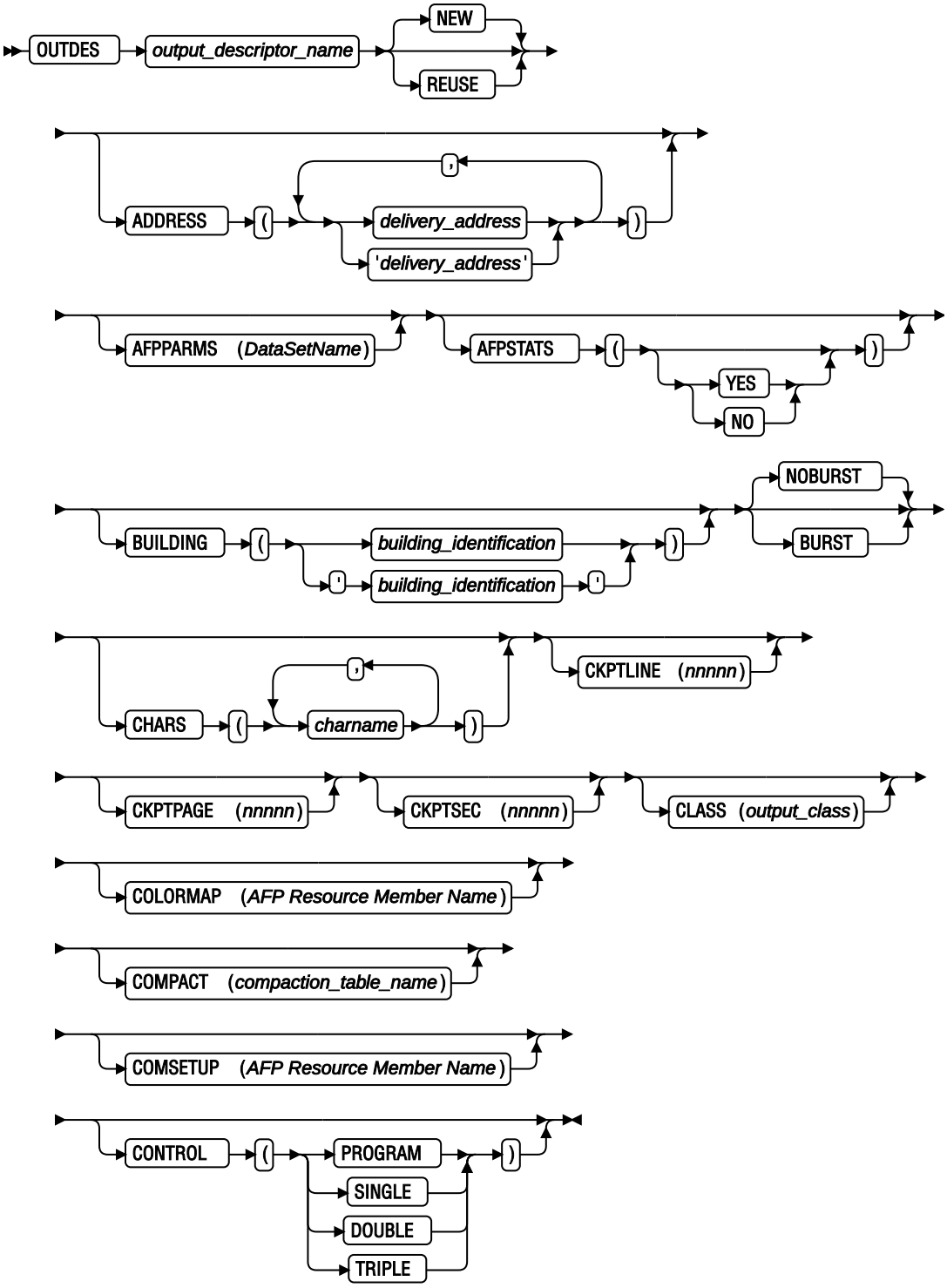
The OUTDES operand of the ALLOCATE command and the PRINTDS command allow you to specify a list of installation-defined output descriptors that were created by OUTPUT JCL statements in the LOGON procedure and by the OUTDES command. You can specify up to 128 output descriptors associated with a SYSOUT data set. See the “ALLOCATE command” on page 8 or the “PRINTDS command” on page 215 for more information.

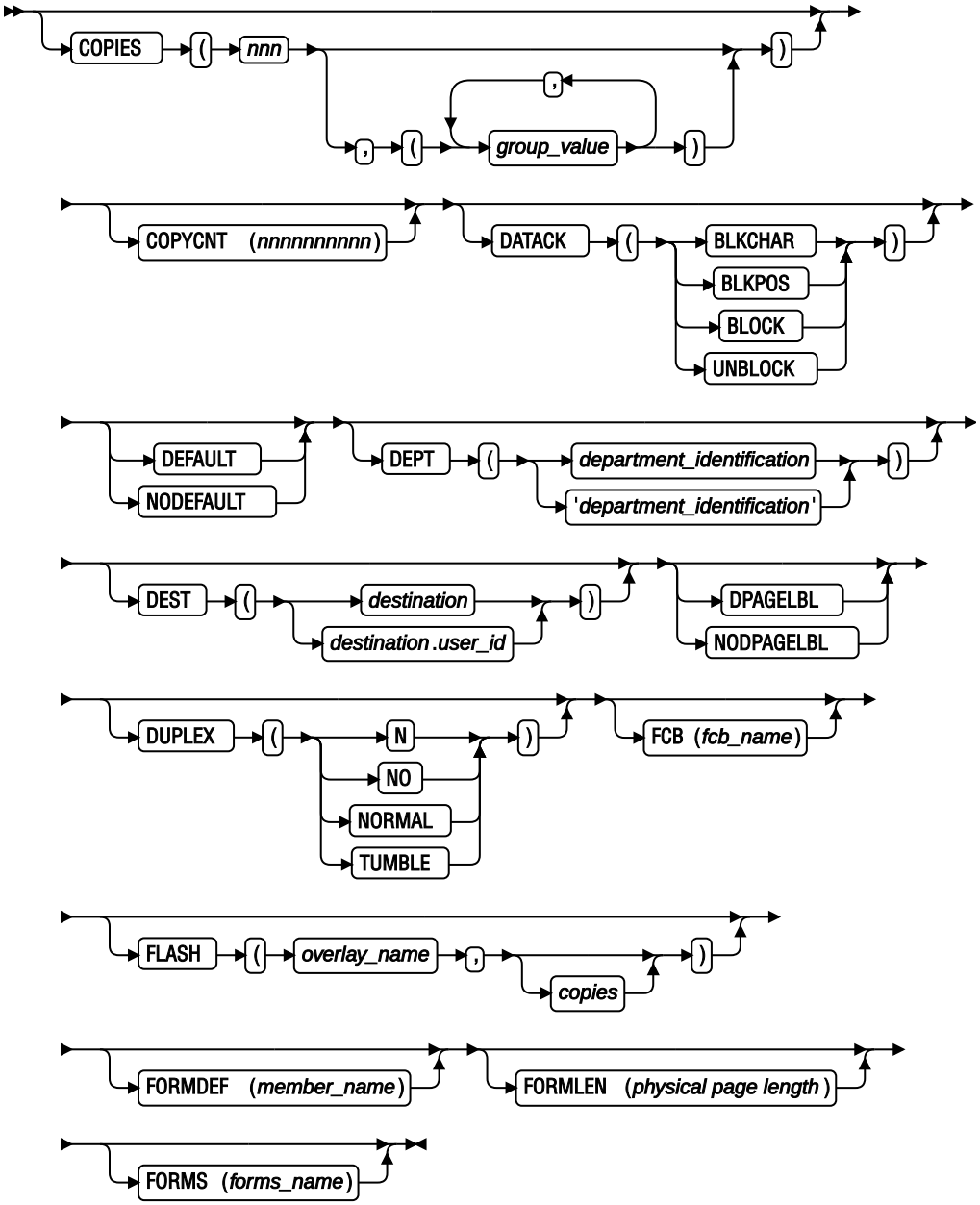
Use operands on the OUTDES command to specify the following information:

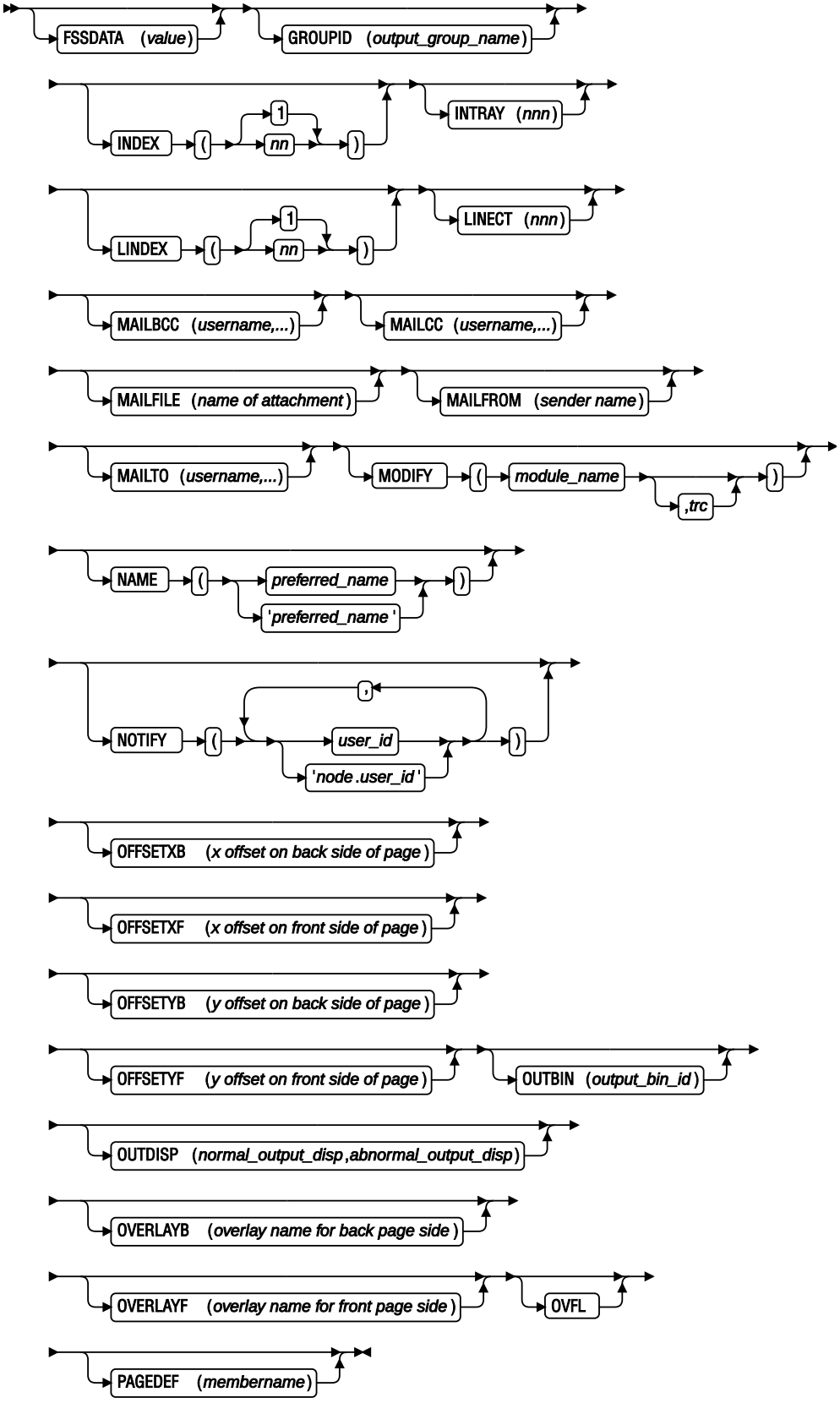
- The name of the output descriptor to be created
- The NEW operand to create the output descriptor. The REUSE operand to replace an existing output descriptor.
- Output characteristics. The format and meanings of the output characteristics are described in *z/OS MVS JCL Reference*.

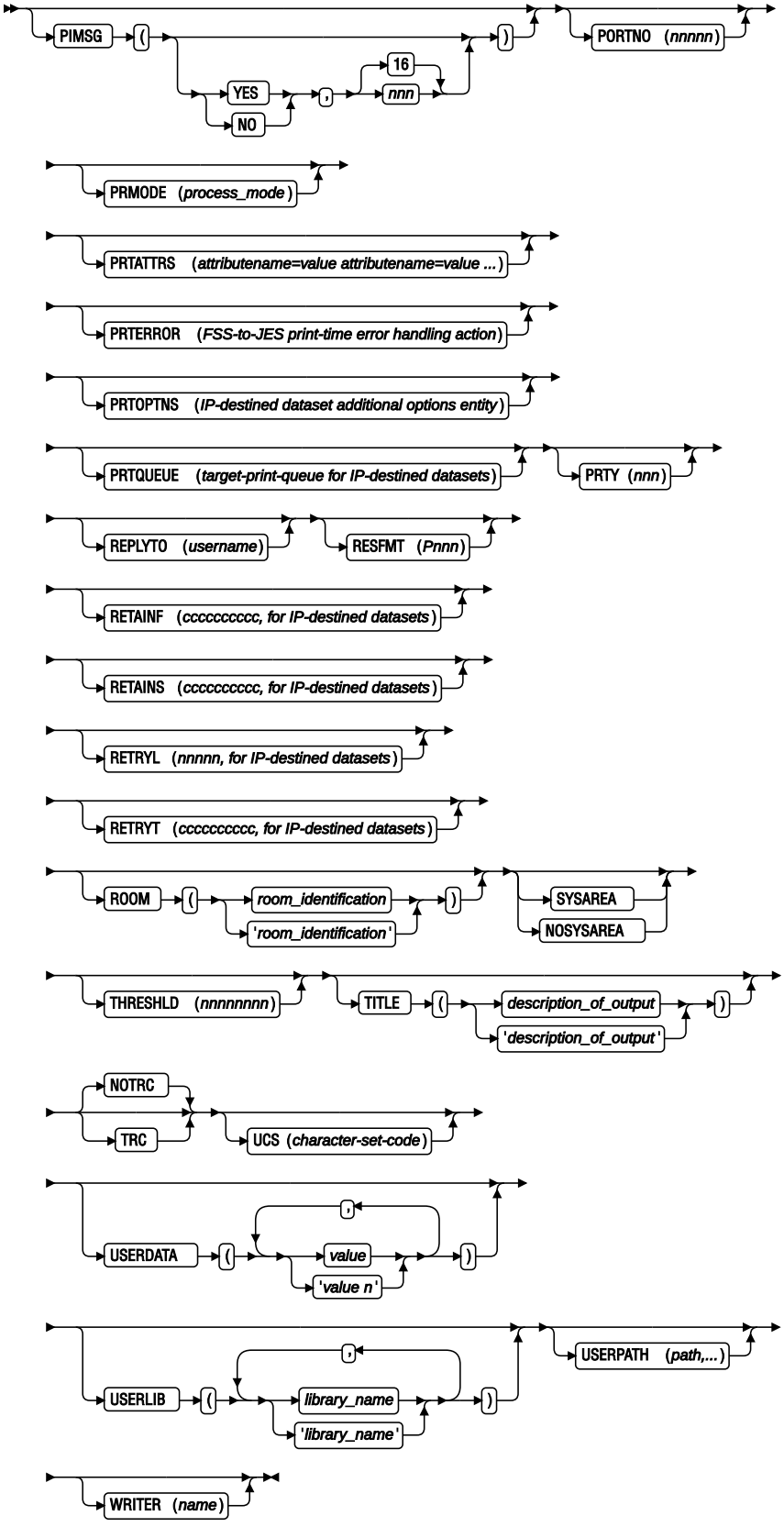
For information about special considerations when using OUTDES, see *z/OS MVS Programming: Authorized Assembler Services Guide*.

OUTDES command syntax




COPIES
(
nnn
)
,
(
group_value
)
COPYCNT (nnnnnnnnn)
DATACK
(
BLKCHAR
BLKPOS
BLOCK
UNBLOCK
)
DEFAULT
NODEFAULT
DEPT
(
department_identification
'department_identification'
)
DEST
(
destination
destination.user_id
)
DPAGELBL
NODPAGELBL
DUPLEX
(
N
NO
NORMAL
TUMBLE
)
FCB (fcb_name)
FLASH
(
overlay_name
,
copies
)
FORMDEF (member_name)
FORMLEN (physical page length)
FORMS (forms_name)


FSSDATA (value)
GROUPID (output_group_name)
INDEX (1 nn)
INTRAY (nnn)
LINDEX (1 nn)
LINECT (nnn)
MAILBCC (username,...)
MAILCC (username,...)
MAILFILE (name of attachment)
MAILFROM (sender name)
MAILTO (username,...)
MODIFY (module_name ,trc)
NAME (preferred_name 'preferred_name ')
NOTIFY (, user_id 'node .user_id ')
OFFSETXB (x offset on back side of page)
OFFSETXF (x offset on front side of page)
OFFSETYB (y offset on back side of page)
OFFSETYF (y offset on front side of page)
OUTBIN (output_bin_id)
OUTDISP (normal_output_disp,abnormal_output_disp)
OVERLAYB (overlay name for back page side)
OVERLAYF (overlay name for front page side)
OVFL
PAGEDEF (membername)


PIMSG
(
)
YES
NO
,
16
nnn
PORTNO (nnnnn)
PRMODE (process_mode)
PRTATTRS (attributename=value attributename=value ...)
PRTERROR (FSS-to-JES print-time error handling action)
PRTOPTNS (IP-destined dataset additional options entity)
PRTQUEUE (target-print-queue for IP-destined datasets)
PRTY (nnn)
REPLYTO (username)
RESFMT (Pnnn)
RETAINF (ccccccccc, for IP-destined datasets)
RETAINS (ccccccccc, for IP-destined datasets)
RETRYL (nnnnn, for IP-destined datasets)
RETRYT (ccccccccc, for IP-destined datasets)
ROOM
room_identification
'room_identification'
SYSAREA
NOSYSAREA
THRESHLD (nnnnnnnn)
TITLE
description_of_output
'description_of_output'
NOTRC
TRC
UCS (character-set-code)
USERDATA
value
'value n'
USERLIB
library_name
'library_name'
USERPATH (path,...)
WRITER (name)

OUTDES command operands

output_descriptor_name
specifies the name of the output descriptor to be created or reused. This operand is required. Specify 1 to 8 alphanumeric characters for the name. The first character must be alphabetic or one of the special characters #, $, or @.

The OUTDES operand of the ALLOCATE, PRINTDS, and FREE commands refers to the *output_descriptor_name* specified.

NEW | REUSE

NEW
specifies that a new output descriptor is to be created. If an output descriptor with the same name exists, the system ends your request unsuccessfully. NEW is the default.

REUSE
specifies that if an output descriptor with the same name is found, the new definition replaces the old one. If an output descriptor with the same name does not exist, OUTDES creates a new output descriptor name.

ADDRESS(*delivery_address*)
specifies the delivery address for system output (SYSOUT). This address prints on the separator pages. You can specify from 1 to 4 delivery addresses. For each delivery address, you can specify from 1 to 60 EBCDIC characters. See "Coding rules" on page 204 for the valid characters allowed with and without quotation marks.

AFPPARMS(*DataSetName*)
Specifies the data set name the AFP Print Distributor uses to locate the parameter file.

AFPSTATS ({YES | NO}))
Produce an AFP Statistics report while printing this sysout data set.

BUILDING(*building_identification*)
specifies the building location associated with the SYSOUT. The building location prints on the separator pages. You can specify from 1 to 60 EBCDIC characters. Refer to "Coding rules" on page 204 for the valid characters allowed with and without quotation marks.

BURST | NOBURST

BURST
specifies that 3800 output is to be burst into separate sheets.

NOBURST
specifies that the printed 3800 output is to be in continuous fanfold pages. NOBURST is the default.

The following parameters are passed on to the scheduler facility, for more information about these parameters see the OUTDES command in *z/OS MVS JCL Reference*.

CHARS(*charname*{,...})
specifies one or more font (character arrangement) tables for printing the SYSOUT data set on a 3800 or 3900 printer. You can specify up to four table names. Specify 1 to 4 alphabetic, numeric, or the special characters #, $, or @ for the character name.

For more information about font (character arrangement) tables, see *IBM 3800 Printing Subsystem Programmer's Guide*.

CKPTLINE(*nnnnn*)
specifies the maximum number of lines contained in a logical page. Specify a value from 0 to 32767. The system uses this value either for JES checkpointing of printed output or for SNA transmission checkpoints. Use CKPTLINE in combination with the CKPTPAGE operand.

If you do not specify CKPTLINE, JES uses an installation default specified at initialization.

CKPTPAGE(*nnnnn*)
specifies the maximum number of pages to be printed or transmitted before the next SYSOUT data set checkpoint occurs. Specify a value from 1 to 32767. This value represents the number of pages to be

transmitted as a single SNA chain when data is transmitted to a SNA workstation. Use CKPTPAGE in combination with the CKPTLINE operand.

If you do not specify CKPTPAGE, JES uses the installation default specified at initialization. The default may also indicate whether checkpoints are to be based on page count or time.

CKPTSEC(*nnnnn*)
specifies the number of seconds that are to elapse between checkpoints of the SYSOUT data set that is printing. Specify a value from 1 to 32767.

If you do not specify CKPTSEC, JES uses the installation default specified at initialization. The default may also indicate whether checkpoints are to be based on page count or time.

CLASS(*output_class*)
assigns the SYSOUT data set to an output class.

output class
A-Z, 0-9, or *, which indicates same output class as MSGCLASS.

COLORMAP(*AFP Resource Member Name*)
specifies the AFP Resource (object) for the data set that contains color translation information.

object
1-8 characters, the first must be either alphabetic or $, #, @; the remaining may be alphanumeric or $, #, @.

COMPACT(*compaction_table_name*)
specifies the name of the compaction table to be used when the data set is transmitted to a workstation. Specify a 1 to 8 alphanumeric character symbolic name. If you do not specify COMPACT, compaction is suppressed for the data set.

COMSETUP(*AFP Resource Member Name*)
Specifies the AFP Resource (object) for the data set that contains setup information.

object
1-8 characters, the first must be either alphabetic or $, #, @; the remaining may be alphanumeric or $, #, @.

CONTROL({PROGRAM | SINGLE | DOUBLE | TRIPLE})
specifies the type of forms control to be used.

PROGRAM
indicates that the carriage control character of each data record is to control line spacing on the form. PROGRAM is the default. The carriage control characters are given in *DFSMS/MVS Macro Instructions for Data Sets*.

SINGLE
indicates forced single spacing.

DOUBLE
indicates forced double spacing.

TRIPLE
indicates forced triple spacing.

COPIES(*nnn*[,(*group_value*)])
specifies the number of copies to be printed for the data set. The number of copies, *nnn*, can range from 1 to 255, subject to an installation limit. The default is 1.

If you use COPIES in a referenced FORMDEF memeber (described later), the system ignores the COPIES value.

If you specify group values, the system ignores the individual value, *nnn*, for the 3800 printer. The group values describe how the printed copies are to be grouped (3800 printer only). Each group value specifies the number of copies of each page that are to be grouped together. You can specify up to 8 group values. For example, a group value of 3 causes the first page of a data set to be printed three times before printing is started for the second page, which might also be printed three times, and so forth.

COPYCNT(*nnnnnnnnnn*)
specifies number of copies to be printed.

nnnnnnnnnn
0-2147483647 (JES2 or JES3)

DATACK({BLKCHAR | BLKPOS | BLOCK | UNBLOCK})
specifies whether "print positioning" and "invalid character" data check errors are to be blocked or unblocked for printers accessing through the functional subsystem Print Services Facility™ (PSF).

BLKCHAR
specifies character errors that are not valid are to be blocked. The errors are not reported to PSF. Print positioning errors are reported normally.

BLKPOS
specifies print positioning errors are to be blocked, and not reported to PSF.

BLOCK
specifies neither print positioning errors nor character errors that are not valid are reported to PSF.

UNBLOCK
specifies both print positioning errors and character errors that are not valid are reported to PSF.

If you do not specify DATACK, the DATACK specification from the PSF PRINTDEV statement is used. If it is not specified in the PRINTDEV statement, the default is BLOCK.

DEFAULT | NODEFAULT

DEFAULT
specifies that the output descriptor defined by this OUTDES command is the default output descriptor. SYSOUT data sets that do not explicitly refer to an output descriptor use the output characteristics specified in this OUTDES command.

NODEFAULT
specifies that an ALLOCATE or PRINTDS command must explicitly reference the output descriptor to use the defined output characteristics specified in this OUTDES command.

Note: When a default output descriptor is defined with a CLASS value, TSO/E commands such as ALLOCATE, PRINTDS, and SMCOPY may use their own default output class instead.

DEPT(*department_identification*)
specifies the department identification associated with the SYSOUT. This department identification prints on the separator pages. You can specify from 1 to 60 EBCDIC characters. See “Coding rules” on page 204 for the valid characters allowed with and without quotation marks.

DEST(*destination* | *destination.user_id*)
specifies the destination of a remote workstation, a user at a specific remote workstation, or an ip-network-address to which the output is routed for processing. You can specify from 1 to 8 characters for either *destination* or *user_id*.

For information about what you can specify for *destination* or *destination.user_id*, see *z/OS MVS JCL Reference*.

DPAGELBL | NODPAGELBL
specify whether the system is to print a security-related character string on each page of output.

DPAGELBL
specifies that the system is to print the character string. The character string is associated with a security label (typically the security label of the user's current session). Your installation determines the character string used.

NODPAGELBL
specifies that the character string is to be suppressed. You must have the appropriate RACF access authority to override page labeling. If you need to override DPAGELBL but are unable to, check your installation security procedures or see your RACF security administrator.

DUPLEX({N | NO | NORMAL | TUMBLE})
specifies whether or not printing is to be done on both sides of the sheet. The DUPLEX keyword overrides the duplex option from the forms definition, if any, specified by the FORMDEF keyword.

N or NO
specifies to print on one side only.

NORMAL
specifies that the physical page is rotated about the Y axis. For most page orientations (including the default orientation), the Y axis is the long edge of the sheet. This allows for binding on the long side of the sheet.

TUMBLE
specifies that the physical page is rotated about the X axis. For most page orientations (including the default orientation), the X axis is the short edge of the sheet. This allows for binding on the short side of the sheet.

If you do not specify DATACK, the DATACK specification from the PSF PRINTDEV statement is used. If it is not specified in the PRINTDEV statement, the default is BLOCK.

FCB(*fcb_name*)
specifies the name of the forms control buffer (FCB) or image to be used for the 3211, 3203-5, or 3800 printers. The name of the FCB is a 1 to 4 alphanumeric character string consisting of the last 1 to 4 characters of the following:

- FCB2xxxx member for the 3211 or 3203-5 printer or printers supported by System Network Architecture (SNA)
- FCB3xxxx member for the 3800 printer.

For more information about the forms control buffer, see:

- *z/OS DFSMSdfp Advanced Services*
- *IBM 3800 Printing Subsystem Programmer's Guide*

FLASH(*overlay_name*[,*copies*])
specifies the name of a forms overlay, which can be used by the 3800 Printing Subsystem. The overlay is "flashed" on a form or other printed information over each page of output. The forms *overlay_name* must be 1 to 4 alphabetic, numeric, or special characters #, $, or @. Optionally, you can specify the number of *copies* on which the overlay is to be printed. The count can range from 0 to 255. To flash no copies, specify a count of zero.

FORMDEF(*member_name*)
specifies the member name of a partitioned data set containing information that the Advanced Function Printer (AFP 3800-3 or 3800-8) uses to print a data set. The member can contain the following information:

- The overlays that are to be invoked during output processing
- The location on the page where the overlays are to be placed
- The suppressions that can be activated for specified page formats.

The member name contains a maximum of 6 characters, of which the first 2 are predefined by your installation. For the last 4 characters, specify alphabetic, numeric, or the special characters #, $, or @.

FORMLEN(*physical page length*)
specifies the numerical length and unit type that PSF will use to change the physical paper length without reconfiguring the printer.

FORMLEN (*nn.nnnUU)*

size
nn.nnn, where n is a digit 0-9. A maximum of two digits to the left of the decimal point and three digits to the right are allowed. The decimal point and the three digits to the right of the decimal point are optional. If a number less than 1 is desired, a 0 must appear to the left of the decimal point (for example, 0.5IN).

units
UU, where UU is either IN for inches or CM for centimeters.

FORMS(*forms_name*)
specifies the name of the form on which the output is to be printed. Specify 1 to 8 alphabetic, numeric, or the special characters #, $, or @ for the forms name.

If you do not specify FORMS, JES uses the installation default specified at initialization.

FSSDATA(*value*)
allows arbitrary values to be passed from a spooling product to an FSA or other despooler.

value
1-127 EBCDIC text characters.

GROUPID(*output_group_name*)
specifies the name to be used by JES to identify which of a job's SYSOUT data sets are to form an output group. The output group name consists of 1 to 8 alphanumeric characters and is selected by the system programmer to define an output group for the job.

INDEX(*nn*)
specifies a value indicating the data set indexing print offset (to the right) for the 3211 printer with the indexing feature. The width of the print line is reduced by the value of INDEX. Specify a value from 1 to 31. The value 1 indicates flush left. The values 2 to 31 indent the print line by *nn*-1 positions.

The default is 1, which indicates flush left.

INTRAY(*nnn*)
Specifies the paper source.

nnn
1-255.

LINDEX(*nn*)
specifies a value indicating the data set indexing print offset (to the left) for the 3211 printer with the indexing feature. The width of the print line is reduced by the value of LINDEX. Specify a value from 1 to 31. The value 1 indicates flush right. The values 2 to 31 move the right margin over by *nn*-1 positions.

The default is 1, which indicates flush right. LINDEX is ignored on printers other than the 3211 printer.

LINECT(*nnn*)
specifies the number of lines that are to be printed before overflow processing. Specify a value from 0 to 255. If you specify zero, no overflow processing is done.

If you do not specify LINECT, JES obtains the value from one of the following:

1. The LINECT field of the accounting information parameter on the JCL JOB statement.
2. The installation default specified at JES initialization.

MAILBCC(*username,...*)
specifies one or more e-mail addresses of the blind secondary e-mail recipients.

1-32 addresses can be specified, each address can be up to 60 characters.

MAILCC(*username,...*)
specifies one or more e-mail addresses of the secondary e-mail recipients.

1 to 32 addresses can be specified, each address can be up to 60 characters.

MAILFILE(*name of attachment*)
specifies the name to use for the attached file when an attachment is included in an e-mail.

The name can be up to 60 characters.

MAILFROM(*sender name*)
specifies a name or other information that identifies who the e-mail is from.

The information can be up to 60 characters.

MAILTO(*username,...*)
specifies one or more e-mail addresses of the primary e-mail recipients.

1 to 32 addresses can be specified, each address can be up to 60 characters.

MODIFY(*module_name*[,*trc*])
specifies the name of a copy modification module, which is loaded into the 3800 or 3900 Printing Subsystem. This module contains predefined data such as legends, column headings, or blanks. The module specifies where and on which copies the data is to be printed. USE IEBIMAGE to define and store the module in the SYS1.IMAGELIB system data set. Specify 1 to 4 alphabetic, numeric, or the special characters #, $, or @ for the *module_name*.

The table reference character (TRC) corresponds to the character set(s) specified on the CHARS operand. Values are 0 for the first table-name, 1 for the second, 2 for the third, or 3 for the fourth.

NAME(*preferred_name*)
specifies the preferred name to be associated with the SYSOUT. The name prints on the separator pages to identify the owner of the SYSOUT. You can specify from 1 to 60 EBCDIC characters. See "Coding rules" on page 204 for the valid characters allowed with and without quotation marks.

NOTIFY({*user_id* | *node.user_id*} ...)
specifies the user ID that is to receive a print completion message. The message identifies the output that has completed printing and indicates whether the printing was successful. You can specify 1 to 4 user IDs to which to send the print completion message.

A JES2 system issues the print complete message when all the SYSOUT data sets for an output group have printed. An output group consists of the SYSOUT data sets printed between the output header page and the output trailer page of a job. A JES3 system issues the print complete message when the SYSOUT data sets for the same printer and the same job have printed.

If you do not specify node, NOTIFY defaults to the node where the job was submitted.

OFFSETXB(*x offset on back side of page*)
specifies the offset in the x direction from the page origin (or partition origin for N_UP) for the back side of each page of output.

offset
mmmm(.nnn)UU where:

mmmm, nnn
decimal digits.

UU
the units either:

- IN for inches, CM for centimeters.
- MM for millimeters, PELS or POINTS.

  If units are PELS or POINTS, a whole number must be specified. (example: 12.345MM or 678PELS)

OFFSETXF(*x offset on front side of page*)
specifies the offset in the x direction from the page origin (or partition origin for N_UP) for the front side of each page of output.

offset
mmmm(.nnn)UU where:

mmmm, nnn
decimal digits.

UU
the units either:

- IN for inches, CM for centimeters.
- MM for millimeters, PELS or POINTS.

If units are PELS or POINTS, a whole number must be specified. (example: 12.345MM or 678PELS)

OFFSETYB(*y offset on back side of page*)
specifies the offset in the y direction from the page origin (or partition origin for N_UP) for the back side of each page of output.

offset
mmmm(.nnn)UU where:

mmmm, nnn
decimal digits.

UU
the units either:

- IN for inches, CM for centimeters.
- MM for millimeters, PELS or POINTS.

  If units are PELS or POINTS, a whole number must be specified. (example: 12.345MM or 678PELS)

OFFSETYF(*y offset on front side of page*)
specifies the offset in the y direction from the page origin (or partition origin for N_UP) for the front side of each page of output.

offset
mmmm(.nnn)UU where:

mmmm, nnn
decimal digits.

UU
the units either:

- IN for inches, CM for centimeters.
- MM for millimeters, PELS or POINTS.

  If units are PELS or POINTS, a whole number must be specified. (example: 12.345MM or 678PELS)

OUTBIN(*output_bin_id*) [4]
specifies the media destination for the SYSOUT data set to be processed by JES2 or by JES3. *output_bin_id* specifies the identifier of the printer output bin on the IBM family of Advanced Function Printers supporting multiple output bins.

The valid range for *output_bin_id* is 1 to 65,535. No default value is provided.

If no OUTBIN operand is given, the Print Services Facility (PSF) will stack the output in the default output bin.

If no *output_bin_id* value is provided with the OUTBIN operand (for example, `OUTDES OUT1 OUTBIN` is entered), the system will prompt you for the required value by issuing the following message:

```
ENTER PRINTER OUTPUT BIN ID
```

If a value for *output_bin_id* is specified that is not one of the supported ones, PSF will stack the output in the printer's default output bin and issue a message indicating that the requested bin is not available.

For more information about multiple media destinations and OUTBIN processing see *PSF/MVS Application Programming Guide*.

OUTDISP(*normal_output_disp*, *abnormal_output_disp*)
specifies the disposition(s) for the output data set for normal and abnormal program terminations.

normal_output_disp
is the disposition for the output data set when the job completes normally. The default for this parameter is WRITE, unless the installation has chosen a different default disposition.

abnormal_output_disp
is the disposition for the output data set when the job completes abnormally. This parameter defaults to the disposition specified in normal-output-disposition, if one was specified. Otherwise, it defaults to the installation default (WRITE).

You can specify one of the following for either or both of the positional parameters:

WRITE
specifies that the output file is to be deleted immediately after processing.

HOLD
specifies that the output data is to be held until released by the TSO/E user or operator. Releasing the output changes its disposition to WRITE.

KEEP
specifies that the output file is to be processed. After processing, the data set disposition changes to LEAVE.

LEAVE
specifies that the output data is to be held until released by the TSO/E user or operator. Releasing the output changes its disposition to KEEP.

PURGE
specifies that the output data set should be deleted before processing.

OVERLAYB(*overlay name for back page side*)
specifies that the named medium overlay is to be placed on the back side of each sheet to be printed.

overlay
1-8 characters, the first must be either alphabetic or $, #, @; the remaining may be alphanumeric or $, #, @.

OVERLAYF(*overlay name for front page side*)
specifies that the named medium overlay is to be placed on the front side of each sheet to be printed.

overlay
1-8 characters, the first must be either alphabetic or $, #, @; the remaining may be alphanumeric or $, #, @.

OVFL()
specifies whether or not JES3 should test for page overflow on an output printer. (JES3 only)

PAGEDEF(*member_name*)
specifies the member of a partitioned data set containing information that the Advanced Function Printer (AFP) uses to print the data set. The member can contain the following information:

- Logical page size and width
- Fonts
- Page segments
- Multiple page types or formats
- Lines within a page; for example, line origin, carriage controls, and spacing
- Multiple logical pages on a physical page.

The member name contains a maximum of 6 characters, of which the first 2 are predefined by your installation. For the last 4 characters, specify alphabetic, numeric, or the special characters #, $, or @.

PIMSG[(YES,*nnn*) | (NO,*nnn*)]
specifies whether messages are to be printed. Values are 0 through 999. The value specifies that the system is to cancel the printing of the current data set after the specified number of errors have been either:

- Detected by the functional subsystem (FSS), or

- Reported to FSS by the printer.

PIMSG(YES)
specifies that messages generated by FSS are to be printed. PIMSG(YES,16) is the default.

PIMSG(NO)
specifies that messages are to be suppressed.

If you specify *nnn* as zero, the system does not cancel the printing of the current data set.

PORTNO(*nnnnn*)
specifies the TCP/IP port number at which the FSS (for example, IP Printway) connects to the printer.

nnnnn
1-65535.

PRMODE(*process_mode*)
specifies the process mode to be used to schedule output data sets either to output devices running under a functional subsystem (FSS) or to an output device managed by JES. For a list of valid process modes, contact your system programmer. If you do not specify PRMODE, JES might determine the process mode based on the content of the data. Specify 1 to 8 alphabetic or numeric characters for the process mode.

Use PRMODE to indicate the type of processing you want for a data set. You can use it to direct JES scheduling of this data set to a particular output FSS or JES writer. You can also use PRMODE to request specific processing of a Network Job Entry (NJE) transmitted data set at the destination node without knowing the device name or a SYSOUT class.

PRTATTRS(*attributename=value attributename=value ...*)
specifies an Infoprint Server job attribute. The Infoprint Server User's Guide documents job attributes names and syntax for acceptable values.

PRTERROR(*FSS-to_JES print-time error handling action*)
specifies how a SYSOUT data set that has had printing terminated by a functional subsystem is to be released to JES.

DEFAULT
indicates the default functional subsystem action is to be taken.

QUIT
indicates the functional subsystem should release the data set as complete even if a terminating error occurred during printing.

HOLD
indicates the functional subsystem should request the data set to be held on the JES spool for possible corrective action if a terminating error occurred.

PRTOPTNS(*IP-destined dataset additional options entity*)
character string of 1 to 16 characters indicating the named entity that contains additional print options for an IP-destined data set that an FSS transmits.

See appropriate functional subsystem documentation for the specific syntax that the FSS supports.

PRTQUEUE(*target-print-queue for IP-destined datasets*)
character string of 1 to 127 characters indicating the target print queue for IP-destined data set that an FSS transmits.

See appropriate functional subsystem documentation for the specific syntax that the FSS supports.

PRTY(*nnn*)
specifies the initial selection priority for the data set. Specify a value from 0 to 255, where 0 is the lowest output processing priority and 255 is the highest output processing priority.

Note: The PRTY(*n*) option will be ignored, unless the JES2 initialization parameter OUTDEF PRTYOUT=YES is also specified. For details see *z/OS JES2 Initialization and Tuning Reference* or contact your JES2 system programmer.

REPLYTO(*username*)
specifies the e-mail address to which recipients of the e-mail should respond.

The address can be up to 60 characters.

RESFMT(*Pnnn*)
specifies the resolution used to format the print data set.

Pnnn
P240 or P300 (pels per inch).

RETAINF(*cccccccccc, for IP-destined datasets*)
character string of 1 to 10 characters indicating how long an FSS will hold on to an IP-destined data set after a failed transmission.

See appropriate functional subsystem documentation for the specific syntax that the FSS supports.

RETAINS(*cccccccccc, for IP-destined datasets*)
character string of 1 to 10 characters indicating how long an FSS will hold on to an IP-destined data set after a successful transmission.

See appropriate functional subsystem documentation for the specific syntax that the FSS supports.

RETRYL(*nnnnn, for IP-destined datasets*)
specifies the number of attempts an FSS will try for transmission of an IP-destined dataset.

nnnnn
0-32767.

See appropriate functional subsystem documentation for the specific syntax that the FSS supports.

RETRYT(*cccccccccc, for IP-destined datasets*)
character string of 1 to 10 characters indicating how much time a functional subsystem will wait between retries of transmission attempts of a data set.

See appropriate functional subsystem documentation for the specific syntax that the FSS supports.

ROOM(*room_identification*)
specifies the room identification to be associated with the output data set. This room identification prints on the separator pages. You can specify from 1 to 60 EBCDIC characters. See “Coding rules” on page 204 for the valid characters allowed with and without quotation marks.

SYSAREA | NOSYSAREA

SYSAREA
specifies that the output should include a system printable area. (See also NOSYSSAREA).

NOSYSAREA
specifies that the output should not include a system printable area. (See also SYSAREA).

THRESHLD*(nnnnnnnn)*
specifies the maximum number of records for the sysout data set. For more information see OUTDES command in the *z/OS MVS JCL Reference*. This applies to JES3 only.

TITLE(*description_of_output*)
specifies the report title to be associated with the output file. This title prints on the separator pages. You can specify from 1 to 60 EBCDIC characters. See “Coding rules” on page 204 for the valid characters allowed with and without quotation marks.

TRC | NOTRC

TRC
specifies whether the data records contain table reference character (TRC) codes. The codes identify the font to be used to print each record.

A TRC code immediately follows the carriage control character, if any. Its value corresponds to either one of the four fonts specified by CHARS or one of the fonts in the PAGEDEF font list. PAGEDEF allows more than four fonts to be specified.

NOTRC
specifies that the data set does not contain TRC codes. NOTRC is the default.

UCS(*character-set-code*)
specifies universal character set, print train, or character-arrangement table for a 3800 Printing Subsystem.

character-set-code
1-4 alphanumeric or $, #, @ characters.

USERDATA(*value*)
specifies the installation-defined values for the installation's prescribed processing. If your installation has defined further keywords through installation exits, that optional processing can be requested on the output descriptor with this keyword. Refer to your installation's definition for the intended use of this keyword operand.

You can code up to 16 installation-defined values for this keyword as previously specified by your installation. Each value can be 1 to 60 EBCDIC text characters. Apostrophes around each value are required if the string contains a blank, comma, tab, or semicolon; apostrophes are optional for all other EBCDIC characters. However, if the string contains an apostrophe, code two apostrophes and enclose the entire string in single apostrophes such as USERDATA(`'USERKEY1=User's value'`). Null positions such as USERDATA(*value_1*`,,`*value_3*) or USERDATA(`,`*value_2*`,`*value_3*) are not allowed.

USERLIB(*library_name* {,*library_name* ...})
specifies the data set name(s) of the libraries that contain the Advanced Function Printer (AFP) resources that the Print Services Facility (PSF) uses when processing the SYSOUT data set. The AFP resources that specify how the PSF is to print the SYSOUT data set are:

- Fonts
- Page Segments
- Overlays
- Pagedefs
- Formdefs

Note: This parameter is not supported for PSF/MVS direct-attached printing.

You can use user libraries to maintain secure resources (such as signatures in private data sets), keep resources that are being tested in a private data set during the test period, or personalize and maintain your own library.

PSF searches for resources first in the resource libraries specified by USERLIB, then in the system-defined resources.

library_name specifies the data set name of a library containing the Advanced Function Printer (AFP) resources. The specified library can contain any AFP resources.

The data set name must follow the rules for MVS data set names. See *z/OS MVS JCL Reference*, for the rules regarding data set names. If the application supports the specification of unqualified data set names and you specify the USERLIB parameter without quotation marks, the specified data set name is concatenated to the system-defined high-level qualifier.

If you do not specify the USERLIB parameter, only the system and installation print resources are available for use.

A library is a partitioned data set (PDS). Member names are the same as the requested resource. When you create a member, the member name should be unique to all libraries in the search concatenation.

When you use the USERLIB keyword:

- You must have read access (for example, via RACF) to libraries specified by USERLIB.
- Libraries must be cataloged in a catalog that is available to PSF/MVS.

- Libraries must be accessible to PSF while processing the SYSOUT data set.
- Library data sets are dynamically deallocated after PSF has processed the SYSOUT data set.

See *PSF/MVS Application Programming Guide* for more information about the USERLIB keyword.

USERPATH(*path,...*)
names HFS or zFS file system paths containing resources to be used by Print Services Facility (PSF) when processing sysout data sets.

USERPATH={path, ... }

- path is the path name only. It cannot include the file name.
- you can omit the parentheses if you code only one path.
- a USERPATH parameter can specify from 1 to 8 path subparameters.
- USERPATH=(,path) is invalid.
- if the path contains any special characters, blanks, or is continued to the next line, it must be enclosed in apostrophes.
- the first character in path is a slash.
- a path can be specified as a maximum of 255 characters including any blank characters.
- see the PATH parameter on the DD statement for additional syntax rules.

WRITER(*external_writer_name*)
specifies a name for use in processing or selecting a SYSOUT data set. If you specify the external writer name, the output data set is written under the control of that external writer rather than the control of JES2 or JES3. The writer name can contain 1 to 8 alphabetic, numeric, or the special characters #, $, or @.

For more information about external writers, see *z/OS MVS Using the Subsystem Interface*.

Coding rules

- The following characters are valid in strings *with* quotation marks:
  - Any valid EBCDIC character.
  - Two consecutive single quotation marks to specify a single quotation mark in a quoted string.
  - Enclose values that contain blanks in quotation marks.
  - A semicolon (;) is allowed within a quoted string unless you are issuing the command under ISPF or PCF. When under ISPF or PCF, the semicolon or the alternate character your installation may have specified as the command delimiter, still functions as a command delimiter and may cause a syntax error.
- The following characters are valid in strings *without* quotation marks:
  - Alphanumeric
  - Special Characters:
    - @ is represented as X'7C'
    - $ is represented as X'5B'
    - # is represented as X'7B'

    Character sets that use hexadecimal representations other than those listed previously might cause an error.

OUTDES command return codes

Table 30 on page 205 lists all the return codes of OUTDES command.

[4] The OUTBIN operand on the TSO/E OUTDES command is the equivalent to the OUTBIN keyword of the JCL OUTPUT statement.

Table 30: OUTDES command return codes

Return codes	Meaning
0	Processing successful.
12	The installation exit requested termination.
16	Processing unsuccessful.

OUTDES command examples

Example 1

This example shows how the OUTDES, ALLOCATE, and FREE commands work together to define, reference, and free the dynamic output descriptor.

Operation: Specify the OUTDES command to define the dynamic output descriptor.

Known:

- Name of the new output descriptor: MULTCOPY
- Number of copies: 3
- Pages are to be burst
- Output class: I

```
outdes multcopy copies(3) burst class(i) new
```

Operation: Specify the ALLOCATE command to reference the dynamic output descriptor.

Known:

- Name of the file: SYSPRINT
- Name of the output descriptor: MULTCOPY

```
allocate file(sysprint) new outdes(multcopy)
```

Operation: Specify the FREE command to free the file and the dynamic output descriptor.

Known:

- Name of the file: SYSPRINT
- Name of the output descriptor: MULTCOPY

```
free file(sysprint) outdes(multcopy)
```

Example 2

Operation: Specify the OUTDES command to define the dynamic output descriptor.

Known:

- Name of the output descriptor: ONECOPY
- Number of copies: 1
- No security labels are to be printed on pages.
- Pages are to be burst.

```
outdes onecopy copies(1) nodpagelbl burst new
```

Example 3

Operation: Specify the OUTDES command to reuse the dynamic output descriptor.

Known:

- Name of the output descriptor: MULTCOPY
- Number of copies: 3
- The output will fill the whole page including the system printable area.
- Replace the existing MULTCOPY output descriptor.

```
outdes multcopy copies(3) nosysarea reuse
```

Example 4

Operation: Specify the OUTDES command to print routing information about the separator pages.

Known:

- Name of the new output descriptor: NEWDEST
- Address for delivery is:
  - IBM Corporation
  - P.O. Box 950
  - Poughkeepsie, NY
  - 12602
- Building to use for distribution: 510
- Data set disposition if the job completes normally: KEEP
- Data set disposition if the job completes abnormally: PURGE
- DEPT to be placed on the report: Payroll
- NAME to be placed on the report: J. Plant
- Room to be placed on the report: Conference Room 'A'
- Title to be placed on the report: OVER-TIME

```
outdes newdest("_")
address('IBM Corporation','P.O. Box 950','Poughkeepsie, NY, 12602')
building(510) outdisp(keep,purge) dept(Payroll)
name('J. Plant') room('Conference Room 'A'')
title(OVER-TIME)
```

Example 5

Operation: Specify the OUTDES command with a default normal disposition and a specified abnormal disposition.

Known:

- Name of the new output descriptor: DESTA
- Default data set disposition if the job completes normally: WRITE
- Data set disposition if the job completes abnormally: PURGE

```
outdes desta outdisp(,purge)
```

Example 6

Operation: Specify the OUTDES command with a specified normal disposition and default abnormal disposition.

Known:

- Name of the new output descriptor: DESTB
- Data set disposition if the job completes normally: PURGE
- Default data set disposition if the job completes abnormally: PURGE

```
outdes destb outdisp(purge)
```

Example 7

Operation: Specify the OUTDES command with specified normal and abnormal dispositions.

Known:

- Name of the new output descriptor: DESTC
- Data set disposition if the job completes normally: PURGE
- Data set disposition if the job completes abnormally: HOLD

```
outdes destc outdisp(purge,hold)
```

Example 8

Operation: Specify the OUTDES command to define a user library for PSF resources.

Known:

- Name of the new output descriptor: NEWDESC
- Page definition name to be used: STNDRD
- Libraries to be searched: USER.PRIVATE.RESOURCE and then GROUP.PRIVATE.RESOURCE

```
outdes newdesc new pagedef(stndrd)("_")
userlib('user.private.resource','group.private.resource')
```

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 workstation.
- Release the output for a job for printing by the subsystem.

OUTPUT is a foreground-initiated-background (FIB) command. This command is generally used in conjunction with SUBMIT, STATUS, and CANCEL commands.

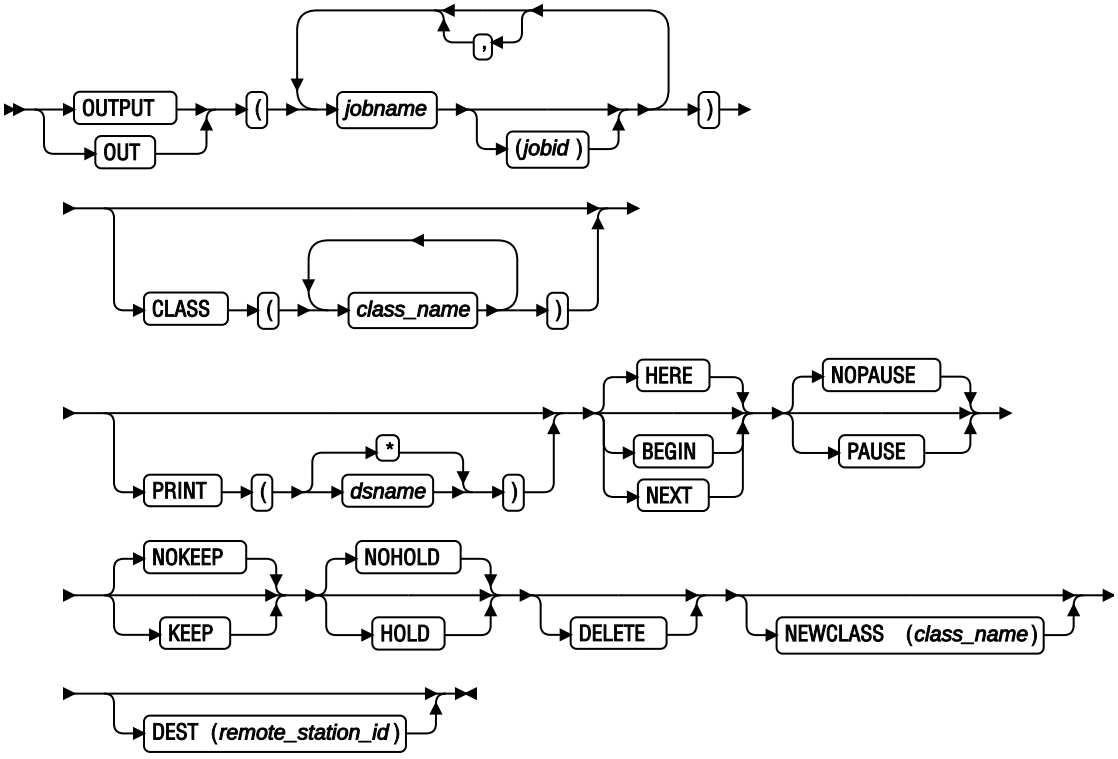
The OUTPUT command applies to all jobs whose job names begin with your user identification. Access to jobs whose job names do not begin with a valid user identification must be provided by an installation-written exit routine. The SUBMIT, STATUS, and CANCEL commands apply to batch jobs. You must have special permission to use these commands.

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 notifies you when the job terminates, giving you the opportunity to use the OUTPUT command. MSGCLASS and SYSOUT data sets should be assigned to reserved classes or explicitly held to be available at the terminal.

If your installation uses security labels and security checking, the output for a job has a security label associated with it. To use the OUTPUT command to process job output, the security label you are logged on with must be equal to or greater than the security label at which the job ran.

Note: You cannot specify both the KEEP and DEST keywords while using the OUTPUT command. These two keywords, when specified together with the OUTPUT command, cause a message to appear stating that the specification is not valid because of conflicting keywords.

OUTPUT command syntax



OUTPUT command operands

(*jobname* (*jobid*))
specifies one or more names of batch or foreground jobs. The job name for foreground session is user ID. Each job name must begin with your user identification and, optionally, can include one or more additional characters unless the default IBM-supplied installation exit that scans and checks the job name and user identification is replaced by a user-written routine. The system processes the held output from the jobs identified by the job name list.

To avoid duplicate job names, you should include the optional job ID for uniqueness. The job ID is a unique job identifier assigned by the job entry subsystem (JES) at the time the job was submitted to the batch system. Currently the only valid forms of job identifiers (*jobid*) assigned by JES are:

- JOBnnnnn or Jnnnnnnn – Jobs
- STCnnnnn or Snnnnnnn – Started Tasks
- TSUnnnnn or Tnnnnnnn – TSO Users

CLASS(*class_name*)
specifies the name or names of the output classes to be searched for output from the jobs identified in the job name list. If you do not specify the name of a class, all held output for the jobs are available. A class name is a single character or digit (A-Z or 0-9).

PRINT(*dsname* | *)
specifies the name of the data set to which the output is to be directed. If unqualified, the prefix is added to and the qualifier OUTLIST is appended to the data set name. You can substitute an asterisk

for the data set name to indicate that the output is to be directed to your terminal. If you omit both the data set name and the asterisk, the default value is the asterisk. PRINT is the default value if you omit PRINT, DELETE, NEWCLASS, DEST, and HOLD/NOHOLD.

If the PRINT data set is not pre-allocated, RECFM defaults to FBS, LRECL defaults to 132, and the BLKSIZE defaults to 3036.

BEGIN | HERE | NEXT

BEGIN
indicates output operations for a data set are to start from the beginning of the data set regardless of whether it has been checkpointed.

HERE
indicates 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 is processed from the beginning. If you omit HERE, BEGIN, and NEXT, then HERE is the default.

NEXT
indicates output operations for a data set that has been previously checkpointed are to be skipped. Processing resumes at the beginning of non-checkpointed data sets.

CAUTION: The checkpointed data sets that are skipped are deleted unless the KEEP operand is specified.

PAUSE | NOPAUSE

PAUSE
indicates output operations are to pause after each SYSOUT data set is listed to allow you to enter a SAVE or CONTINUE subcommand. Pressing the Enter key after the pause causes normal processing to continue. This operand can be overridden by the NOPAUSE operand of the CONTINUE subcommand. If PAUSE is not specified, then NOPAUSE is the default.

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

KEEP | NOKEEP

KEEP
specifies the SYSOUT data set is to remain enqueued after printing (see also HOLD and NOHOLD).

NOKEEP
specifies the SYSOUT data set is to be deleted after it is printed. If neither KEEP nor NOKEEP is specified, then NOKEEP is the default.

HOLD | NOHOLD

HOLD
specifies the kept SYSOUT data set is to be held for later access from the terminal.

Note: HOLD may be overridden if DEST(*remote_station_id*) specifies a network job entry (NJE) node. For example,

```
TSO OUTPUT job DEST(DETROIT) HOLD
```

issued on a node in TAMPA will not hold the output.

For JES3 users, HOLD may also be overridden if NEWCLASS(*class_name*) specifies a class defined on a JES3 SYSOUT initialization statement with a default NJE networking node DEST. For example,

```
SYSOUT,CLASS=D,TYPE=PRINT,DEST=DETROIT
```

is included in the JES3 initialization stream.

```
TSO OUTPUT job NEWCLASS(D) HOLD
```

issued on a node in TAMPA will not hold the output.

Note to JES3 Users: To view the output, you must specify an output class that has been defined as HOLD (for TSO/E) or RSVD (reserved) on the DD statement. If you specify RSVD class, then MSGCLASS and SYSOUT class must be the same as the RSVD class. For more information, see *z/OS JES3 Initialization and Tuning Guide*.

NOHOLD
specifies the kept SYSOUT data set be released for printing by the subsystem. NOHOLD is the default.

DELETE
specifies classes of output specified with the CLASS operand are to be deleted.

NEWCLASS(*class_name*)
is used to change one or more SYSOUT classes to the class specified by the *class_name* subfield.

DEST(*remote_station_id*)
routes SYSOUT classes to a remote workstation specified by the station ID subfield. The station ID is 1 to 8 characters in length.

Output sequence

Output is 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),
//             DISP=(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
//             DISP=OLD
//SYSIN        DD           *
             SCRATCH VTOC,VOL=3330=333000
/*
```

To retrieve the output, you enter:

```
output (jwsd581 jwsd582) class (x y z)
```

Your output is displayed in the following order:

1. Output of job JWSD581
   a. class X (JCL and messages)
   b. class Y (SYSPRINT data)
   c. class Z (SYSUT2 data)
2. Output of job JWSD582
   a. class X (JCL and messages)
   b. class Y (SYSPRINT data)
   c. message (NO CLASS Z OUTPUT FOR JOB JWSD582)

If no classes are specified, the jobs are processed as entered. Class sequence is not predictable.

Subcommands for the OUTPUT command

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.

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. You might see records repeated or notice records missing if you attempt 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 is 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 is deleted.
- The attention key is pressed during the printing of a data set and the END subcommand is entered.

OUTPUT command return codes

Table 31 on page 211 lists the return codes of OUTPUT command.

Table 31: OUTPUT command return codes

Return codes	Meaning
0	Processing successful.
12	Processing unsuccessful. An error message has been issued.

OUTPUT command examples

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” on page 212).

```
out kean1 class(s) delete
```

Example 5

Operation: Retrieve SYSOUT data from your session at your terminal.

Known:

- The TSO/E user ID: 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)
```

OUTPUT subcommands (overview)

The subcommands of the OUTPUT command are:

Table 32: Subcommands and functions of the OUTPUT command

Subcommand	Function
CONTINUE	Resumes output operations that have been interrupted.
END	Ends the OUTPUT command.
HELP	Obtains the syntax and function of the OUTPUT subcommands.
SAVE	Copies the SYSOUT data set from the spool to the named data set.

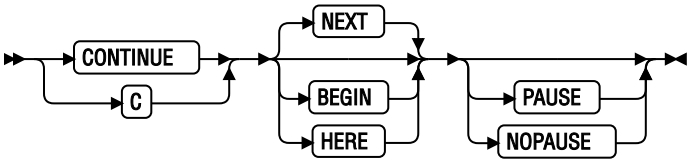
OUTPUT—CONTINUE subcommand

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.

OUTPUT—CONTINUE subcommand syntax



OUTPUT—CONTINUE subcommand operands

BEGIN
indicates output operations are to be resumed from the beginning of the data set being processed at the time of interruption.

HERE
indicates output operations are to be resumed at a point of interruption. If the attention key is pressed, processing resumes at the approximate point of interruption in the current data set. If end-of-data is reached and PAUSE is specified, processing resumes at the beginning of the next data set (even if it is checkpointed and HERE is specified on the command).

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 is specified on the command, processing starts at the beginning of the next data set. If HERE is specified, processing starts at the checkpoint of the next data set or at its beginning, if no checkpoint exists. If NEXT is specified, processing starts at the beginning of the next non-checkpointed data set. If BEGIN, HERE, and NEXT are omitted, then NEXT is the default.

Note: The interrupted and skipped data set, or both are deleted unless you specified KEEP on the OUTPUT command.

PAUSE
indicates output operations are to pause after each data set is processed to allow you to enter a SAVE subcommand. Pressing the Enter key after the pause causes normal processing to continue. You can use this operand to override a previous NOPAUSE condition for output.

NOPAUSE
indicates output operations are not to be interrupted. You can use this operand to override a previous condition for output.

OUTPUT—CONTINUE subcommand examples

Example 1

Operation: Continue output operation with the next SYSOUT data set.

```
continue
```

Example 2

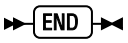
Operation: Start output operations over again with the current data set being processed.

```
continue begin
```

OUTPUT—END subcommand

Use the END subcommand to terminate the operation of the OUTPUT command.

OUTPUT—END subcommand syntax



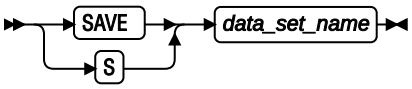
OUTPUT—HELP subcommand

Use the HELP subcommand to obtain the syntax and function of the OUTPUT subcommands. For a description of the HELP command syntax and function, see the “HELP command” on page 143.

OUTPUT—SAVE subcommand

Use the SAVE subcommand to copy the SYSOUT data set from the spool data set to the named data set. If you use the data set with the PRINT operand, then it must be a valid data set. There is no restriction against saving JCL. To use SAVE, you should specify the PAUSE operand on the OUTPUT command. SAVE does not save the entire SYSOUT output of the job, only the data set currently being processed.

OUTPUT—SAVE subcommand syntax



OUTPUT—SAVE subcommand operand

data_set_name
specifies the new data set name to which the SYSOUT data set is to be copied.

OUTPUT—SAVE subcommand examples

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
```

PRINTDS command

Use the PRINTDS command to format and print data sets on any printer defined to the Job Entry System (JES). PRINTDS allows you to:

- Print data sets that have the following characteristics:
  - Sequential or partitioned (print the entire data set or selected members) Different types of control characters cannot be mixed within a sequential data set or in a PDS. However, different members of a PDS can contain different types of control characters. For more information about control characters, see "PRINTDS command operands" on page 217.
  - Movable or unmovable.
  - Fixed or variable record format.
  - Logical record length not greater than 32,760.
  - Resides on DASD.

- Reference output descriptors.
- Format the data and either print it or copy it to a data set.
- Print data sets that contain Document Composition Facility (DCF) data.

Note: Generation data group (GDG) data sets are not supported by PRINTDS.

There are three types of operands you can specify on the PRINTDS command:

- The name and characteristics of the data set(s) or file to be printed.
- The formatting and output characteristic operands.
- The OUTDES operand referring to a previous output descriptor.

Process for the input data set or file

Each data set you specify is processed as follows:

- If you specify a file, the data sets within the file concatenation are allocated and printed separately. They are treated as if you had specified a list of data sets to be printed. After the system prints the file, it does not deallocate the file.
- PRINTDS examines the first line of the data set to determine whether Document Composition Facility (DCF) formatted the data in the data set. If so, the first line is of the following form and PRINTDS extracts device and font information from it. examines the first line of the data set and extracts the device and font information from that line, such as:

  ```
  SCRIPT/VS Rx.x.x;  DEVICE device CHARS font1 (… font4)
  ```

  If PRINTDS finds page mode data in the data set, the device and font information will not be extracted.

  PRINTDS associates the specified font information with the SYSOUT data set. If you specify the CHARS operand on the PRINTDS command, the system uses the values specified on the CHARS operand when it allocates the SYSOUT data set rather than the values from the DCF data set.

Output for a data set or file

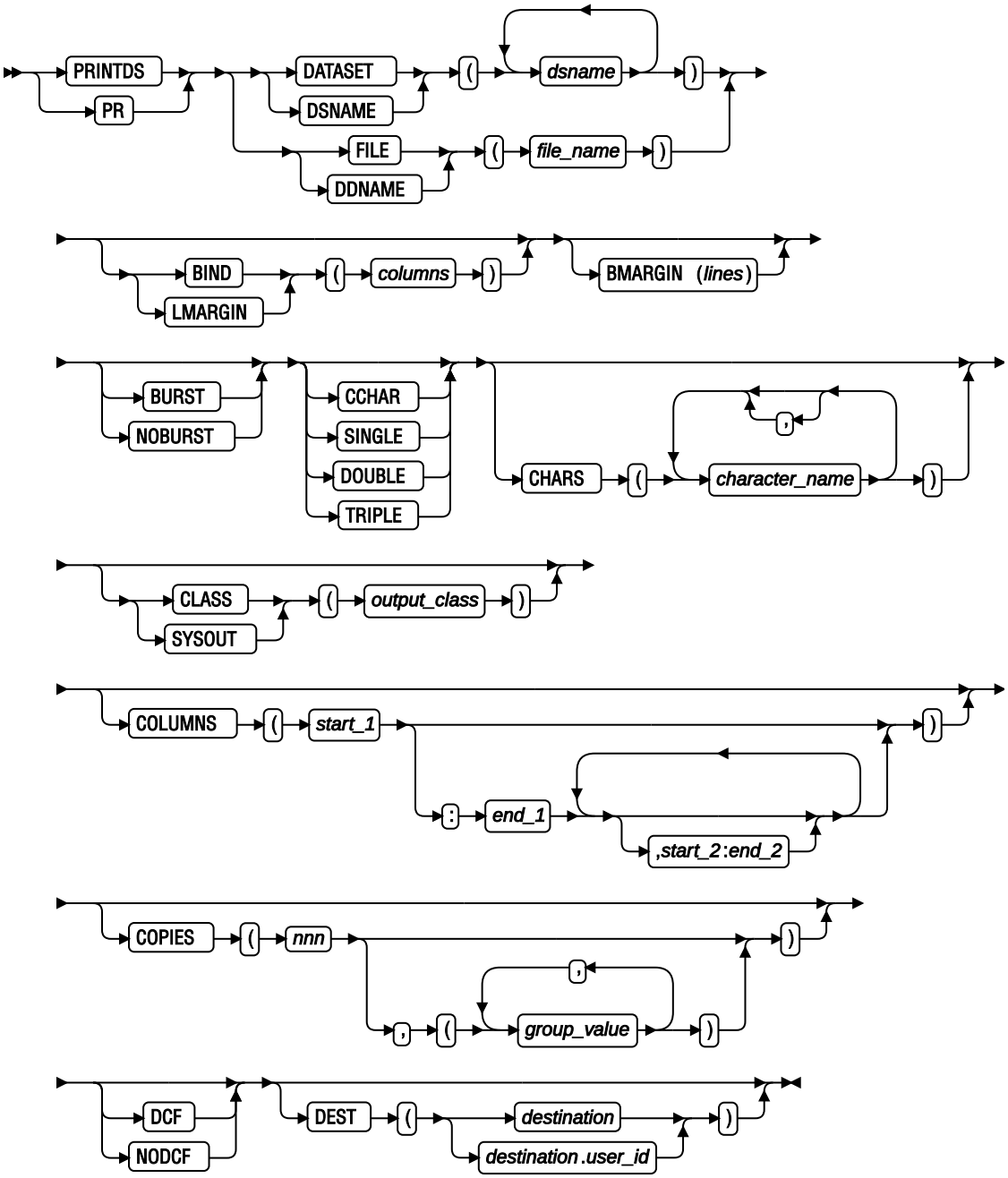
The system prints a data set or file using the formatting operands you specified. It prints a title that contains the name of the data set and the page number on every page, unless the NOTITLE operand is specified or defaulted.

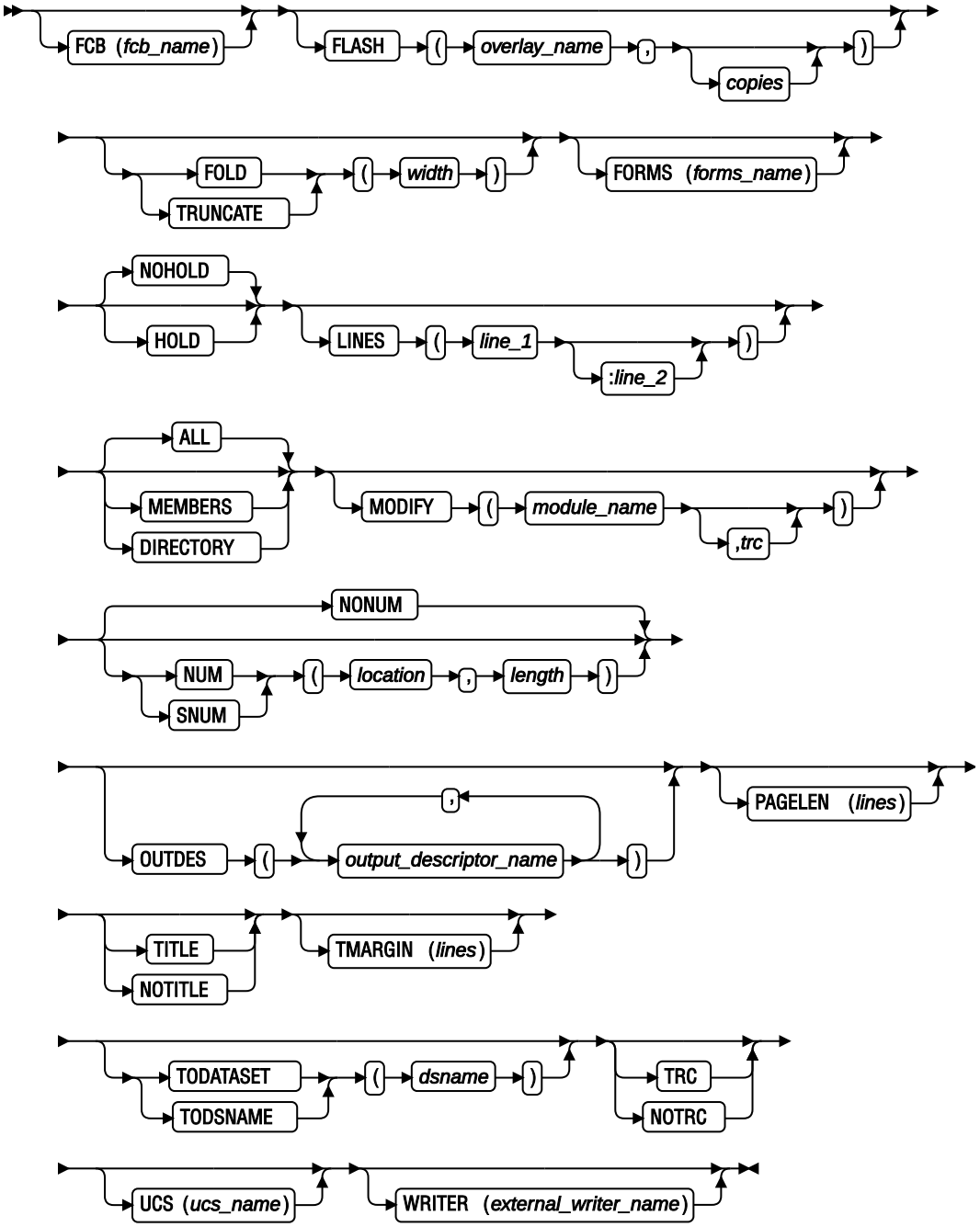
If the output attributes of a member are different from the previous member, such as a Document Composition Facility file, then the system prints it separately. If you specify multiple input data sets or members with unlike attributes, then the system creates more than one SYSOUT data set. If you specify more than one output descriptor using the OUTDES operand, then the system also creates more than one SYSOUT data set.

If you direct the output to a data set, the system does not allocate SYSOUT data sets. The formatted output is placed in the data set specified on the TODATASET operand. If the member or data set does not exist, PRINTDS creates it.

Introductory information about how to use the PRINTDS command is described in *z/OS TSO/E User's Guide*.

PRINTDS command syntax





PRINTDS command operands

DATASET(*dsname*) | DSNAME(*dsname*)

specifies either one or more data sets or members to be printed. If you do not specify DATASET, DSNAME, FILE, or DDNAME, the system prompts you to enter the name. The data set name must include the descriptive (rightmost) qualifier and can contain a member name in parentheses.

If the data set is password protected, suffix the data set name with a slash (/) and the password.

The data set must have a data set organization of PO or POU for partitioned or partitioned unmovable, PS or PSU for sequential or sequential unmovable, record formats of fixed or variable, and logical record length not greater than 32,760.

Spanned records or records with track overflow are not supported. You can specify up to 255 data sets.

Either DATASET, or DSNAME, or FILE, or DDNAME is required. If you do not specify FILE, DDNAME, DATASET, or DSNAME, the system prompts you to enter the name.

FILE(*file_name*) | DDNAME(*file_name*)
specifies the name of the file to be printed. The data sets within the concatenation are printed as if you had specified the DATASET operand followed by the list of the data set names that make up the file.

You cannot use the FILE operand to print a data set that is protected by a READ password. Use the DATASET or DSNAME operand.

Either FILE, or DDNAME, or DATASET, or DSNAME is required. If you do not specify FILE, DDNAME, DATASET, or DSNAME, the system prompts you to enter the name.

BIND(*columns*) | LMARGIN(*columns*)
specifies the number of columns that the output is to be shifted to the right. LMARGIN is an alias for BIND. You can specify between 0 to 255 columns. If you print a partitioned data set, the BIND or LMARGIN value applies only when members are printed. The system ignores the BIND value when the directory portion of the partitioned data set is printed.

Do not use LMARGIN with page mode data. An error occurs if LMARGIN is specified with page mode data.

BIND(0) is the default. A nonzero BIND value is mutually exclusive with the DIRECTORY operand.

BMARGIN(*lines*)
specifies the number of blank lines to be left at the bottom of each printed page. You can specify a minimum of 0 lines, and a maximum of 6 lines less than the value specified or defaulted for the PAGELEN operand.

BMARGIN(0) is the default. A nonzero BMARGIN value is mutually exclusive with the CCHAR or DIRECTORY operand.

BURST | NOBURST
specifies whether 3800 output is to be bursted into separate sheets. BURST or NOBURST is allowed only when you print data to a SYSOUT data set. Therefore, you cannot specify BURST or NOBURST when you specify the TODATASET operand.

NOBURST
specifies that the printed output is to be in continuous fanfold pages.

NOBURST
is the default for a SYSOUT data set.

CCHAR | SINGLE | DOUBLE | TRIPLE

CCHAR
specifies that ANSI or machine code spacing control characters existing in the data set are to be used for inter-record spacing. If you specify CCHAR, the system assumes the default of NOTITLE.

SINGLE
specifies that all non-blank lines from the input data set are to be printed with single spacing.

DOUBLE
specifies that all non-blank lines from the input data set are to be printed with double spacing.

TRIPLE
specifies that all non-blank lines from the input data set are to be printed with triple spacing. If you specify SINGLE, DOUBLE, or TRIPLE, the system ignores blank lines from the input data set.

If you specify CCHAR, SINGLE, DOUBLE, or TRIPLE, the record format recorded in the data set's DSCB is not used to determine the carriage control type in the input. Instead, the first character in the first record of each input data set or member is examined to determine the type of carriage control. If it is a valid machine carriage control character, then the entire data set or member is assumed to have machine carriage control spacing. Otherwise, ANSI carriage control spacing is assumed.

If you do not specify CCHAR, SINGLE, DOUBLE, or TRIPLE, the record format recorded in the data set's DSCB indicates whether the data set contains carriage control characters, and if so, the type.

If you do not specify CCHAR, PRINTDS determines the type of data set (ANSI or MCC) from the DSCB.

If you specify CCHAR, PRINTDS checks if the data set contains a valid MCC code. If it does not find a valid MCC, PRINTDS treats the data set as an ANSI type data set.

Do not specify SINGLE, DOUBLE, or TRIPLE for an input data set that contains ANSI or machine carriage control characters because the inter-record spacing for such a data set is under control of the carriage control characters within the data set.

If you use the COLUMNS, NUM, or SNUM operands with CCHAR, column 1 refers to the first character after the carriage control character. If you specify the TRC operand, then column 1 is the first character after the table reference character.

Table 33 on page 219 contains the valid machine printer carriage control characters.

Table 33: Valid machine printer carriage control characters

Print line and then act	Action	Act immediately and do not print
--	NOOP (Comment line, no print)	X'03'
X'01'	Print only (no space)	--
X'09'	Space 1 line	X'0B'
X'11'	Space 2 lines	X'13'
X'19'	Space 3 lines	X'1B'
X'89'	Skip to channel 1	X'8B'
X'91'	Skip to channel 2	X'93'
X'99'	Skip to channel 3	X'9B'
X'A1'	Skip to channel 4	X'A3'
X'A9'	Skip to channel 5	X'AB'
X'B1'	Skip to channel 6	X'B3'
X'B9'	Skip to channel 7	X'BB'
X'C1'	Skip to channel 8	X'C3'
X'C9'	Skip to channel 9	X'CB'
X'D1'	Skip to channel 10	X'D3'
X'D9'	Skip to channel 11	X'DB'
X'E1'	Skip to channel 12	X'E3'
X'5A'	Defines page mode line of data	--

CHARS(*character_name*{,...})
specifies the name of the character arrangement table (font). You can specify up to four fonts. Specify 1 to 4 alphabetic, numeric, or special characters #, $, or @ for the font. If you specify CHARS, the system assumes the TRC operand, not the default of NOTRC.

Note: To define a single font to be used to print a data set that contains no TRC codes, specify CHARS. To prevent the system from interpreting the first character of each printed line as a TRC code, also specify NOTRC.

CLASS(*output_class*) | SYSOUT(*output_class*)
specifies the output class JES is to use for processing the specified data set. Valid output classes are characters A-Z or 0-9. The default output class is A. SYSOUT is an alias for CLASS.

COLUMNS(*start_1*[: *end_1*[,*start_2*: *end_2*,...]]
specifies the columns of the data set to printed. You can specify the columns as pairs of numbers in the format *start:end*. If you do not specify *end*, the system assumes the last column of the input as end. You can specify up to 32 column pairs.

If your input data set contains a carriage control character or a table reference character (TRC), column 1 refers to the first character position after the carriage control character or the table reference character.

COPIES(*nnn*[,(*group_value*,...)])
specifies the number of copies to be printed for the data set. The number of copies, *nnn*, can range from 1 to 255, subject to an installation limit.

If you specify group values, the system ignores the individual value, *nnn*, for the 3800 printer. The group values describe how the printed copies are to be grouped (3800 printer only). Each group value specifies the number of copies of each page that are to be grouped together. You can specify up to 8 group values. For example, a group value of 3 causes the first page of a data set to be printed three times before printing is started for the second page, which might also be printed three times, and so forth.

COPIES(1) is the default value for a SYSOUT data set.

DCF | NODCF
specifies whether the font information is to be extracted from the first line of a DCF formatted data set. For example,

```
SCRIPT/VS Rx.x.x;  DEVICE device CHARS font1 (… font4)
```

The system finds and uses the font information when the data set is printed. If it is page mode data, the device and font information is not extracted.

NODCF specifies that the font information is not to be extracted from the data set.

If you specify DCF, the system assumes NOTITLE. If you specify DCF and the data set is found to have been formatted by DCF, then machine carriage control spacing is also assumed. However, if you specify DCF and the data set is not formatted by DCF, the system ignores the DCF operand. DCF is the default for a SYSOUT data set.

If you specify DCF and the FILE operand, the first line of each data set within the file concatenation is examined for the DCF information. The data sets making up the file are processed as if you had specified a list of separate data sets.

If DCF is specified or defaulted and the first record of the data set indicates that the data has been formatted by DCF for a 1403 printer, the system assumes NOTRC unless you specified TRC. In all other cases, DCF data sets are assumed to have been formatted with TRC characters unless you had explicitly specified NOTRC.

Note: If you specify DCF, the input data set might not have been formatted by the Document Composition Facility. PRINTDS checks only the first record to determine whether the data set should be processed as a DCF data set. If you specify NODCF, PRINTDS does not check the data set.

DEST{*destination* | *destination.user_id*}
specifies the destination of a remote workstation or a user at a specific remote workstation to which the output is routed for processing. You can specify from 1 to 8 characters for either *destination* or *user_id*.

For information about the destination format systemname.printername, see *z/OS JES2 Initialization and Tuning Guide*.

Or, if you specified a default destination in the SYS1.UADS data set, the DEST output descriptor overrides the destination in SYS1.UADS. See *z/OS JES3 Initialization and Tuning Guide*.

FCB(*fcb_name*)
specifies the name of the forms control buffer (FCB) image to be used for the 3211, 3203-5, 3262, 4248, 6262, or 3800 printers. The name of the FCB is a 1 to 4 alphanumeric character string consisting of the last characters of the following:

- FCB2xxxx member for the channel attached line printers (3203, 3211, 3262, 4245. 4248, 6262) or printers supported by System Network Architecture (SNA)
- FCB3xxxx member for the 3800 printer.
- FCB4xxx member for the 3262, 4248 or 6262 printer

Your installation supplies a default for the SYSOUT class or for the printers.

FLASH(*overlay_name*[,*copies*])
specifies the name of a forms overlay, which can be used by the 3800 Printing Subsystem. The overlay is "flashed" on a form or other printed information over each page of output. The forms *overlay_name* must be 1 to 4 alphabetic, numeric, or special characters #, $, or @. Optionally, you can specify the number of *copies* on which the overlay is to be printed. The count can range from 0 to 255. To flash no copies, specify a count of zero.

FOLD(*width*) | TRUNCATE(*width*)
specifies the length of the printed line if the input line is longer than the output line.

FOLD
specifies that *width* is the maximum length of the output line. Records that are too long to be printed within that length are wrapped around onto subsequent lines.

TRUNCATE
specifies that *width* is the maximum length of the output line. Records that are too long to be printed within that length are truncated to fit on one line.

If the input data set contains carriage control characters, the data being folded or truncated begins after the carriage control character. If the input data set has a table reference character, or a carriage control character and table reference character, the data being folded or truncated begins after the table reference character.

FORMS(*forms_name*)
specifies the name of the form on which the output is to be printed. Specify 1 to 4 alphabetic, numeric, or the special characters #, $, or @ for the forms name.

HOLD | NOHOLD

HOLD
specifies whether the output is to be held in the JES held output queue. NOHOLD specifies that the output be made available for printing immediately.

NOHOLD
is the default for a SYSOUT data set.

LINES(*line_number_1*[: *line_number_2*])
specifies the range of lines to be printed, either in:

- Embedded line number fields using the NUM or SNUM operand, or
- Relative records using the NONUM operand.

If you specify the first line number value only, printing continues from that line to the last line of the data set. Only lines with line number values within the specified range are printed. For example, `LINES(10:20)` causes the 10th through 20th lines of the data set to be printed. However, if the data set has at least 10 lines, but fewer than 20 lines, all lines from the 10th to the end of the data set are printed. If the data set has fewer than 10 lines, no lines are printed.

The line number values you specify for LINES are used for each printed data set. For example, `LINES(1:10)` prints the first 10 lines of every sequential data set and member specified. It also prints the first 10 lines of each member for every partitioned data set specified.

MEMBERS | DIRECTORY | ALL
specifies which portion of a partitioned data set is to be printed.

MEMBERS
specifies that the system is to print only the data contained in the members of the indicated partitioned data set, without the directory. The system prints the members in alphabetical order.

DIRECTORY
specifies that the system is to print only the directory.

ALL
specifies that the system is to print both the data contained in the members and the directory. The members are printed first followed by the directory. ALL is the default.

If you specify MEMBERS, DIRECTORY, or ALL when printing a sequential data set or a specific member of a partitioned data set, the system ignores these operands. If you print a partitioned data set with the ALL operand, you can specify certain operands that are normally *not* allowed when you specify DIRECTORY. The following operands affect the formatting and printing of members of partitioned data sets, but not the directory:

- BIND
- COLUMNS
- DCF or NODCF
- FOLD or TRUNCATE
- LINES
- NUM or SNUM or NONUM
- SINGLE or DOUBLE or TRIPLE
- BMARGIN
- TMARGIN
- NOTITLE

The output of each page of a partitioned data set directory contains the following:

- Two directory lines
- A blank line
- A directory header line
- Another blank line
- One or more lines of directory information

Each directory page has at least 6 lines, unless the partitioned data set has no members. If the partitioned data set has no members, only the directory title lines and header line are printed.

If you specify NOTITLE with the ALL operand, the members of the partitioned data set and other sequential data sets are printed without title lines. However, the directory portion of the partitioned data set is printed with the directory title lines on each page.

MODIFY(*module_name*[,*trc*])
specifies the name of a copy modification module, which is loaded into the 3800 or 3900 Printing Subsystem. This module contains predefined data such as legends, column headings, or blanks. The module specifies where and on which copies the data is to be printed. Use the IEBIMAGE utility to define and store the module in the SYS1.IMAGELIB system data set. Specify 1 to 4 alphabetic, numeric, or the special characters #, $, or @ for the *module_name*.

The table reference character (TRC) corresponds to the character set(s) specified on the CHARS operand. Values are from 0 to 3.

NUM(*location*,*length*) | SNUM(*location*,*length*) | NONUM
specifies where line numbers are in the data set and whether PRINTDS is to print the line numbers.

NUM
indicates that the data set contains a line number field to be printed. The location value is the column location of the beginning of the line number field. The length value is the number of

columns that the line number field occupies. You can specify up to 8 for the length value. Both the location value and the length value are required.

SNUM
indicates the data set contains a line number, but the line number is *not* to be printed. The location value is the column location of the beginning of the line number field. The length value is the number of columns that the line number field occupies. You can specify up to 8 for the length value. Both the location value and the length value are required.

If you specify either NUM or SNUM, the line number field in each record of the input data set must contain only valid decimal digits, 0 to 9. If the line number field contains characters other than 0 to 9, printing of the data set ends. If you are printing a list of data sets, printing continues with the next data set. If you are printing members of a partitioned data set, printing continues with the next member.

NONUM
indicates that PRINTDS is to treat records as though there are no embedded line numbers. NONUM is the default.

If the input data set records contain a carriage control character or table reference character, the column location refers to the first character after the carriage control character or table reference character.

OUTDES(*output_descriptor_name*[, ...])
specifies a list of installation-defined output descriptors that were created by OUTPUT JCL statements in the LOGON procedure or by the TSO/E OUTDES command. The characteristics of each output descriptor are associated with a SYSOUT data set. Specifying OUTDES eliminates the need to supply information related to the printer or the type of printing to be done. You can specify up to 128 output descriptors. Specify 1 to 8 alphanumeric characters for the name. The first character must be alphabetic or one of the special characters #, $, or @.

If you specify operands with an output descriptor, such as BURST, CHARS, COPIES, and DEST, you can override them by specifying the corresponding operand with PRINTDS. For example, specify the following command:

```
PRINTDS DA(ABC) OUTDES(OUTPR1) NOBURST COPIES(1) DEST(NODEB.USR)
```

The COPIES, NOBURST, and DEST operands override the values specified on the output descriptor.

If you specified a default destination in the SYS1.UADS data set, the DEST output descriptor overrides the destination in SYS1.UADS.

PAGELEN(*lines*)
specifies the number of lines to be printed on a page. The lines value must be from 6 to 4095. The default value is 60. The PAGELEN value less the TMARGIN and BMARGIN must be greater than or equal to 6:

```
                                TMARGIN value
                                |
                                |
                                |
PAGELEN value must be greater   Must have 6 or more lines
than or equal to 6              |
                                |
                                |
                                |
                                BMARGIN value
```

Note: PAGELEN specifies the length of a printed page in terms of the number of lines per logical page. The specified value does not override the maximum lines per physical page that the printing program is using. However, if the value specified is greater than the maximum lines per physical page that the printing program is using, then any remainder from the specified value is printed on the next physical page until the specified value is reached, which ends a physical page.

If you are printing a directory of a partitioned data set, the system uses the number of lines specified in PAGELEN for each page of the directory. It ignores the values specified for TMARGIN or BMARGIN.

For more information about printing a directory of a partitioned data set, see the description for the DIRECTORY/MEMBERS/ALL operand.

TITLE | NOTITLE
specifies that a title, including the name of the data set is to be printed and the page number is to appear on every page of the printed output. NOTITLE specifies that the title is to be suppressed.

TITLE is the default for data sets with *no* carriage control characters. NOTITLE is the default for data sets with carriage control characters. If you specify the CCHAR, TRC, and DCF operands, the default is also NOTITLE.

You cannot specify NOTITLE with the DIRECTORY operand because directory title lines are always printed on directory pages. If you specify NOTITLE to print a partitioned data set with the ALL operand, no title lines appear when the system prints each member. However, the directory pages continue to be formatted with directory title lines to distinguish the directory from the members of the data set.

If you specify TITLE and the input data set contains carriage control characters, the system ignores TITLE and uses NOTITLE to print the data set. However, if a list of input data sets is being printed, the system uses TITLE to print subsequent data sets that do not contain carriage control characters. For example, suppose the data set SEPT87.REPORT is a pre-formatted report that contains carriage control characters. The data set SEPT85.DATA does not contain carriage control characters. If you specify the following command:

```
PRINTDS DA('SEPT87.REPORT' 'SEPT85.DATA') TITLE
```

The system uses NOTITLE for the first data set because it assumes that any title information has already been added to the formatted data set. However, the system uses TITLE for the second data set.

TMARGIN(*lines*)
specifies the number of blank lines to be left at the top of each printed page. You can specify a minimum of 0 lines, and a maximum of 6 lines less than the value specified or defaulted for the PAGELEN operand.

TMARGIN(0) is the default. A nonzero TMARGIN value is mutually exclusive with the CCHAR or DIRECTORY operand.

TODATASET(*dsname*) | TODSNAME(*dsname*)
specifies the name of the data set into which the formatted input data is to be copied. If you specify TODATASET or TODSNAME, a SYSOUT data set is not created.

If the specified data set does not exist, PRINTDS creates the data set. Otherwise, PRINTDS uses the existing data set. If the specified output data set already exists, the output from the PRINTDS command replaces any existing data.

If you specify TODATASET that already exists and the data set is not large enough to hold all of the output, the system issues an error message to inform you to preallocate the data set with more space and to reissue the PRINTDS command.

TRC | NOTRC
specifies whether the data records contain table reference character (TRC) codes. The codes identify the font to be used to print each record. A TRC code immediately follows the carriage control character, if any. Its value corresponds to one of the four fonts specified by CHARS. If you specify TRC, the system assumes NOTITLE.

NOTRC specifies that the data set does not contain TRC codes. NOTRC is the default unless you specify CHARS or DCF. If you specify CHARS or DCF or use the default of DCF and the data set is not formatted for the 1403 printer, the system assumes TRC. On a 1403 printer, the system uses NOTRC.

If you specify COLUMNS, NUM, or SNUM operands with TRC, column 1 refers to the first character after the table reference character.

UCS(*universal_character_set_name*)
specifies the alphanumeric value for the universal character set name. Specify up to 4 characters. If you do not specify the CHARS operand, the system uses the UCS as the default.

WRITER(*external_writer_name*)
specifies a name for use in processing or selecting a SYSOUT data set. If you specify the external writer name, the output data set is written under the control of that external writer rather than the control of JES2 or JES3. The writer name can contain 1 to 8 alphabetic, numeric, or the special characters #, $, or @.

For JES3, you can code the DEST=nodename parameter in the output descriptor with the WRITER=name parameter. However, do not code DEST=nodename.userid in the output descriptor with WRITER=namebecause WRITER=name overrides the specification of DEST=nodename.userid.

Default values for PRINTDS

Table 35 shows a summary of default values for the PRINTDS command. SYSOUT operand defaults apply only when printing to a SYSOUT data set.

Changing these default values for the PRINTDS command is discussed in *z/OS TSO/E Customization*.

Table 34: Summary of default values for the PRINTDS command

Operand	SYSOUT only	Default value	Allowed values
BIND	No	0	0 - 255
TMARGIN	No	0	0 - 4094
BMARGIN	No	0	0 - 4094
PAGELEN	No	60	6 - 4095
CLASS or SYSOUT	Yes	A	A - Z, 0 - 9
BURST or NOBURST	Yes	NOBURST	BURST or NOBURST
COPIES	Yes	1	1 - 255
HOLD or NOHOLD	Yes	NOHOLD	HOLD or NOHOLD
MEMBERS or DIRECTORY or ALL	No	ALL	MEMBERS or DIRECTORY or ALL
NUM or SNUM or NONUM	No	NONUM	NUM or SNUM or NONUM
TITLE or NOTITLE	No	If possible, TITLE, Otherwise, NOTITLE. (See note later in this section.)	TITLE or NOTITLE

Note: If you did not specify the CCHAR, DCF, or TRC operands and the data set does not contain carriage control characters, the default is TITLE. Otherwise, PRINTDS assumes NOTITLE. If NOTITLE is the default, PRINTDS does not print title lines when printing a sequential data set or members of a partitioned data set. However, the directory of the partitioned data set is always printed with title lines, even when you specify NOTITLE.

The destination value used for a SYSOUT data set can be defined by the following statements:

- PRINTDS DEST keyword.
- OUTDES DEST keyword (or output JCL in a TSO/E proc).
  - OUTDES statement referenced is specified by the PRINTDS OUTDES keyword

  - Output JCL statements in a TSO/E proc can be used for a PRINTDS SYSOUT data set if output JCL is an applicable default type
- Default destination (as specified by ACCOUNT DEST keyword) for the user in SYS1.UADS.
- For JES3 only, the SYSOUT initialization statement DEST keyword.

When JES processes the SYSOUT, it incorporates the information in the order listed below.

1. PRINTDS DEST keyword is used if it is given. If no PRINTDS DEST keyword is given, then check if the OUTPUT statement is applicable. The OUTDES keyword on PRINTDS provides the OUTPUT statement.
2. DEST keyword in the OUTPUT statement is used if it is given. If no OUTPUT DEST keyword is given, then check if SYS1.UADS has a default destination defined (DEST keyword).
3. Default destination (DEST keyword) in SYS1.UADS is used if it is given. If there is no default destination (DEST keyword) in SYS1.UADS then check the SYSOUT CLASS for JES3 installation.
4. DEST keyword in SYSOUT CLASS is used for JES3 installation if the SYSOUT CLASS for the PRINTDS contains the DEST keyword.
5. If none of the preceding applied, use JES defaults.

Mutually exclusive operands on PRINTDS

Table 36 shows the mutually exclusive operands on the PRINTDS command:

Table 35: Mutually exclusive operands on the PRINTDS command

You cannot specify this operand	**with these operands:**
CCHAR	BMARGIN, DIRECTORY, PAGELEN, TITLE, TMARGIN
DCF	DIRECTORY, SINGLE or DOUBLE or TRIPLE, TITLE, TODATASET or TODSNAME
DIRECTORY	BIND, BMARGIN, CCHAR or SINGLE or DOUBLE or TRIPLE, COLUMNS, DCF or NODCF, FOLD or TRUNCATE, LINES, NUM or SNUM or NONUM, NOTITLE, TMARGIN
TRC	TITLE, TODATASET or TODSNAME
TODATASET or TODSNAME	BURST or NOBURST, CHARS, CLASS, COPIES, DCF or NODCF, DEST, FCB, FLASH, FORMS, HOLD or NOHOLD, MODIFY, OUTDES, TRC or NOTRC, UCS, WRITER
SNUM	COLUMNS

BMARGIN, TMARGIN, and BIND allow a minimum value of 0. Because specifying a value of 0 for any of these operands is the same as *not* specifying them, BMARGIN(0) and TMARGIN(0) are not considered to be mutually exclusive with CCHAR or DIRECTORY. Likewise, BIND(0) is not mutually exclusive with DIRECTORY. The system ignores the operands.

PRINTDS command return codes

Table 36 on page 226 lists all the return codes of PRINTDS command.

Table 36: PRINTDS command return codes

Return codes	**Meaning**
0	Processing successful.
4	Processing completed, but a warning message has been issued.

Table 36: PRINTDS command return codes (continued)	
Return codes	**Meaning**
8	The input, output, or SYSOUT data set can not be used.
12	An error occurred during the processing of the PRINTDS command.
16	The installation exit requested termination of the PRINTDS command.

PRINTDS command examples

Example 1

Operation: Print all the members of a partitioned data set, but not the directory. Lines longer than 72 characters are to be folded onto more than one line.

Known:

- The name of the data set: JCL.CNTL

```
printds dsname(jcl.cntl) members fold(72)
```

Example 2

Operation: Send the first 250 lines of a sequential data set to a JES held output queue.

Known:

- The name of the data set: NAMES.TEXT

```
printds dataset(names.text) lines(1:250) hold
```

Example 3

Operation: Print a member of a partitioned data set using an output descriptor that is installation-defined.

Known:

- The name of the data set: FOIL.TEXT
- The name of the member: STATUS
- The name of the output descriptor: FOILOUT

```
printds da(foil.text(status)) outdes(foilout)
```

Example 4

Operation: Print a member of a partitioned data set that is also a Document Composition Facility file using the fonts GT10 and GB10.

Known:

- The name of the data set: MEMO.TEXT
- The name of the member: NOTICE
- The first line in the member reads: SCRIPT/VS R2.0: DEVICE 3800N6 CHARS GT10 GB10

```
printds ds(memo.text(notice))
```

Example 5

Operation: Print a member of a partitioned data set that is also a Document Composition Facility file using the character arrangement tables GT12 and GT15.

Known:

- The name of the data set: DOCUMENT.TEXT
- The name of the member: APPROVAL
- The first line in the member reads: `SCRIPT/VS R2.0: DEVICE 3800N6 CHARS ST10 ST12`

```
printds da(document.text(approval)) chars(gt12,gt15)
```

In the preceding example, the fonts GT12 and GT15 for the CHARS operand override the DCF font names ST10 and ST12.

Example 6

Operation: Print the data sets concatenated to a file.

Known:

- The name of the file: SYSPROC

```
printds fi(sysproc)
```

The members of each of the three partitioned data sets are printed followed by the data set directory. By using the FILE operand, you do not have to know the names of the data sets. The system prints them as if you had specified something like the following:

```
printds da('sys1.tso.clist','tools.clist','my.clist')
```

PROFILE command

Use the PROFILE command or the PROFILE subcommand of EDIT to establish, change, or list your user profile. The information in your profile tells the system how you want to use your terminal. You can:

- Define a character-deletion or line-deletion control character (on some terminals)
- Specify whether prompting is to occur
- Specify the frequency of prompting under the EDIT command
- Specify whether you want to accept messages from other terminal users
- Specify whether you want the opportunity to obtain additional information about messages from a CLIST
- Specify whether you want message numbers for diagnostic messages displayed at your terminal
- Specify primary and secondary languages to be used in displaying translated information
- Specify whether variables in the CLIST or authorized REXX variable pools can use storage above the 16MB line

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

Initially, a user profile is prepared for you when arrangements are made for you to use the system. The authorized system programmer creates your user ID and your user profile. The system programmer is restricted to defining the same user profile for every user ID that the programmer creates. This typical user profile is defined when a user profile table (UPT) is initialized to hexadecimal zeroes for any new user ID. Thus, your initial user profile is made up of the default values of the operands discussed under this command. The system defaults, shown in Table 37 on page 229, provide for the character-delete and the line-delete control characters, depending upon what type of terminal is involved:

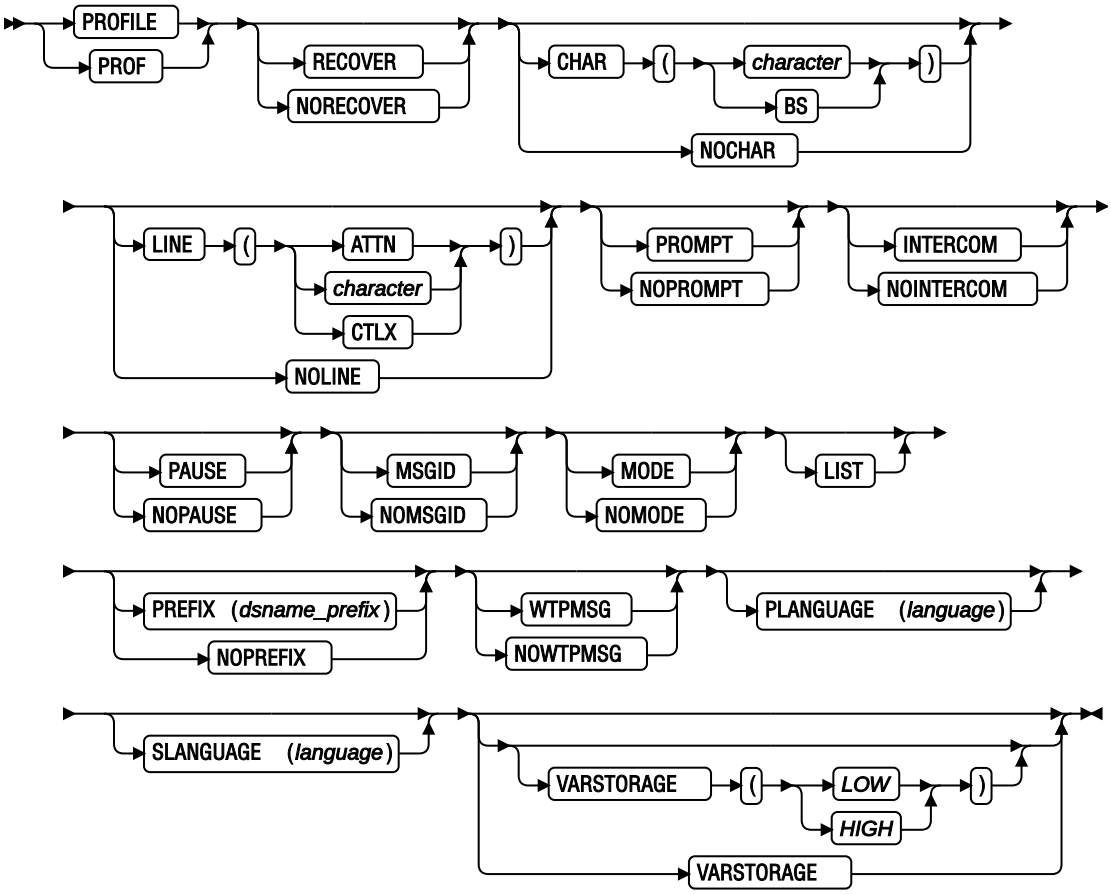
Table 37: System defaults for control characters		
TSO/E terminal	**Character-delete control character**	**Line-delete control character**
IBM 2741 Communication Terminal	BS (backspace)	ATTN (attention)
IBM 3270 Information Display System	None	None
IBM 3290 Information Panel	None	None
IBM 3767 Communication Terminal	None	None
IBM 3770 Data Communication System	None	None

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

Change your profile by using the PROFILE command with the appropriate operands. Only the characteristics that you specify explicitly by operands are changed. Other characteristics remain unchanged. The new characteristics remain valid from session to session. If PROFILE changes do not remain from session to session, your installation might have a LOGON pre-prompt exit that is preventing the saving of any changes in the UPT. Verify this with your system programmer.

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

PROFILE command syntax



PROFILE command operands

RECOVER | NORECOVER

RECOVER
specifies that you can use the recover option of the EDIT command.

Note: You must be able to allocate the two data sets named userid.EDITUTL1 and userid.EDITUTL2, or have them pre-allocated for you in order to use EDIT with your profile set to RECOVER. The high-level qualifier for these two data set names can only be your userid, which might not be the same as your dsname-prefix (specified as a parameter of the PROFILE PREFIX command).

NORECOVER
specifies that you cannot use the recover option of the EDIT command. This is the default value for your profile when the profile is created.

CHAR(*character* | BS) | NOCHAR

CHAR(*character*) [5]
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.

If you are running under Session Manager, the system ignores the 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 try 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 is the character-delete control character.

CHAR(BS) [5]
specifies a backspace signals that the previous character entered should be deleted. This is the default value when your user profile is created.

NOCHAR [5]
specifies no control character is to be used for character deletion.

LINE(ATTN | *character* | CTLX) | NOLINE

LINE(ATTN) [5]
specifies an attention interruption is to be interpreted as a line-deletion control character. This is the default value when your user profile is created.

Note: If a not valid character- and line-delete control character, or both are entered on the PROFILE command, an error message informs you of which specific control character is not valid. The character or line delete field in the user profile table is not changed. You can continue to use the old character- or line-delete control characters.

LINE(*character*) [5]
specifies a control character that you want to use to tell the system to delete the current line. If you are running under Session Manager, the system ignores the control character.

LINE(CTLX) [5]
specifies the X and CCTRL keys (pressed together) on a Teletype terminal are to be interpreted as a line-deletion control character. If you are operating a Teletype terminal, LINE is the default value when your user profile is created.

NOLINE [5]
specifies no line-deletion control character (including ATTN) is recognized.

[5] Not supported with terminals that use VTAM.

PROMPT | NOPROMPT

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

NOPROMPT
specifies no prompting is to occur.

INTERCOM | NOINTERCOM

INTERCOM
specifies that you can receive messages from other terminal users. This is the default value when your user profile is created.

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

PAUSE | NOPAUSE

PAUSE
specifies that you want the opportunity to obtain additional information when a message is issued at your terminal while a CLIST (see the EXEC command) or an in-storage command list (created by using the STACK macro) is executing. After a message that has additional levels of information is issued, the system displays the word PAUSE and waits for you to enter a question mark (?) or press the Enter key.

NOPAUSE
specifies that you do not want to be prompted for a question mark or Enter. This is the default value when your user profile is created.

MSGID | NOMSGID

MSGID
specifies diagnostic messages are to include message identifiers.

NOMSGID
specifies diagnostic messages are not to include message identifiers. This is the default value when your user profile is created.

MODE | NOMODE

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

NOMODE
specifies, when this mode is in effect, the mode message (E or EDIT) is to be issued after a SAVE, RENUM, or RUN subcommand is issued and also when changing from input to edit mode. Specifying PROFILE NOMODE eliminates some of the edit mode messages. NOMODE has the same effect in the background as it does in the foreground. Your profile can be changed by using the PROFILE command with the appropriate operands. Only those characteristics specifically denoted by the operands specified are changed. All other characteristics remain unchanged.

LIST
specifies 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 are changed first, and then the new profile is listed.

After a new user ID is created and before the character-delete and line-delete control character, or both are changed, entering PROFILE LIST results in CHAR(0) and LINE(0) being listed. This indicates the terminal defaults for character-delete and line-delete control characters are used.

Although you receive RECOVER/NORECOVER as an option for this operand, you must be authorized to use the RECOVER options.

PREFIX(*dsname_prefix*) | NOPREFIX

PREFIX(*dsname_prefix*)
specifies a prefix that is to be appended to all non-fully-qualified data set names. The prefix is composed of 1 to 8 alphanumeric characters and begins with an alphabetic character or one of the special characters #, $, or @.

NOPREFIX
specifies no prefixing of data set names by any qualifier is to be performed.

Note: For background processing, the default is the user ID.

WTPMSG | NOWTPMSG

WTPMSG
specifies that you want to receive all write-to-programmer messages at your terminal. Write-to-programmer messages are created by the WTO macro with ROUTCDE=11.

NOWTPMSG
specifies that you do not want to receive write-to-programmer messages. This is the default value when your user profile is created.

PLANGUAGE(*language*)
specifies the primary language to be used in displaying translated information (messages, help information, and the TRANSMIT full-screen panel). You can specify either a 3-character language code or a symbolic language name defined by your installation. If the language name contains one or more blanks, you must enclose the name in quotation marks. See your system administrator for a list of valid language codes and installation-defined language names.

SLANGUAGE(*language*)
specifies the secondary language to be used in displaying translated information should the primary language fail. You can specify either a 3-character language code or a symbolic language name defined by your installation. If the language name contains one or more blanks, you must enclose the name in quotation marks. See your system administrator for a list of valid language codes and installation-defined language names.

VARSTORAGE
specifies the storage location to be used for CLIST variables or REXX OUTTRAP variables containing output from authorized commands. A CLIST or REXX exec uses the VARSTORAGE setting of the PROFILE command when the exec starts. This setting then remains unchanged for the life of the CLIST or REXX exec, even if the CLIST or REXX exec issues a new PROFILE command with a different VARSTORAGE setting. The new setting will only apply when a new CLIST or REXX exec begins.

VARSTORAGE (HIGH)
indicates that CLIST variables and REXX OUTTRAP variables containing output from authorized commands invoked by REXX can be kept in storage above the 16M line.

VARSTORAGE (LOW)
indicates that CLIST variables and REXX OUTTRAP variables containing output from authorized commands invoked by REXX can only be kept in storage below the 16M line. If you specify VARSTORAGE with no operands, VARSTORAGE(LOW) is the default. This is the default value when your user profile is created.

PROFILE language setting notes

If you change your language and then log off, the new language specified may not be saved from session to session. This depends on how your installation defines languages. See your system administrator for assistance.

PROFILE foreground/background processing differences

The following differences should be noted for foreground/background processing:

- Changes made while processing in the foreground are saved from session to session.

- Changes made while processing in the background remain in effect during the background session, and are not saved after the background session. Your foreground profile is not altered by background processing.

See Table 38 on page 233 for a guide to the initialization of the terminal monitor program (TMP) in batch processing. The heading "RACF/Non-RACF Job Without User ID" means RACF without user ID, without a UADS entry for the user ID, or without RACF.

Table 38: UPT/PSCB initialization table in the background.

TMP initialization in the background					
User profile table (UPT)			**Protected step control block (PSCB)**		
	RACF job with USER ID	**RACF/Non-RACF job without USER ID**		**RACF job with USER ID**	**RACF/Non-RACF job without USER ID**
USERFLD	*	ZERO	PSCBUSER and PSCBUID8	job user ID	NULL (blanks) (1) NO JCL
EDIT RECOV	*$	NO RECOVER	PSCBGPNM	NULL	NULL (blanks)
PROMPT	*$	NO PROMPT	OPERATOR	*	NOOPER
MSGID	*	MSGID	ACCOUNT	*	ACCOUNT (1) NO ACCOUNT
INTERCOM	*	NO INTERCOM	JCL	*	JCL
PAUSE	*	NO PAUSE	MOUNT	*	NO MOUNT
ATTN/LD	*	NOT ATTN	ATTN/LD	*	NOT ATTN
MODEMSG	*	NO MODEMSG	EDIT RECOV	*	NO RECOVER
WTPMSG	*	NO WTPMSG	HOLDCLASS	*	NULL (zero)
CHAR DEL	*$	ZERO	SUBMIT CLASS	*	NULL (zero)
LINE DEL	*$	ZERO	SUBMIT MSGCLASS	*	NULL (zero)
PREFIX	1 * 2 job user ID	NULL (blanks) (1) ***	SYSOUT CLASS	*	NULL (zero)
PLANGUAGE	**	** (1) ENU (English)	SYSOUT DEST	*	NULL (blanks)
SLANGUAGE	**	** (1) ENU (English)	CHAR DEL	*	NULL (zero)
			LINE DEL	*	NULL (zero)
			REGION SIZE	*/2	NULL (zero)
VARSTORAGE	*$	LOW			

Table 38: UPT/PSCB initialization table in the background. (continued)

TMP initialization in the background					
User profile table (UPT)			Protected step control block (PSCB)		
	RACF job with USER ID	RACF/Non-RACF job without USER ID		RACF job with USER ID	RACF/Non-RACF job without USER ID
***** The value is taken from UADS entry profile. If the UADS prefix is empty, the system uses the job user ID. ***$** You can modify most of the preceding defaults in the background by issuing the PROFILE command with the appropriate operand/keyword. You cannot use the PROFILE command to modify the attributes in the background. ****** This depends on how your installation defines languages. ******* The value is set equal to the user ID associated with this address space unless that user ID is greater than eight characters in length. In that case, there is no prefix. The search order for the user ID is ACEEUSRI, ASXBUSER, no prefix.			***** The value is taken from the UADS entry profile.		
(1) Setting as a result of using TSO/E Environment Service (IKJTSOEV)					

PROFILE command return codes

Table 39 on page 234 lists the return codes of PROFILE command.

Table 39: PROFILE command return codes

Return codes	Meaning
0	Processing successful.
12	Processing unsuccessful. An error message has been issued.

PROFILE command examples

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 CLIST 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.

```
profile 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(!)
```

If you type:

```
now is the tiâ!bcg!;
```

and press the Enter key, you actually enter:

```
abc.
```

Example 4

Operation: Suppose that you want to receive TSO/E information in Japanese instead of U.S. English.

Known:

- The installation-defined name for Japanese: JAPAN

```
profile planguage(japan)
```

Note: The Japanese language must be active on your system for this command to work.

Example 5

Operation: Suppose that you want to receive TSO/E information in French should the primary language fail.

Known:

- The 3-character language code for French: FRA

```
profile slanguage(fra)
```

Note: The French language must be active on your system for this command to work.

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 *z/OS DFSMS Access Method Services Commands*.

The PROTECT command establishes or changes:

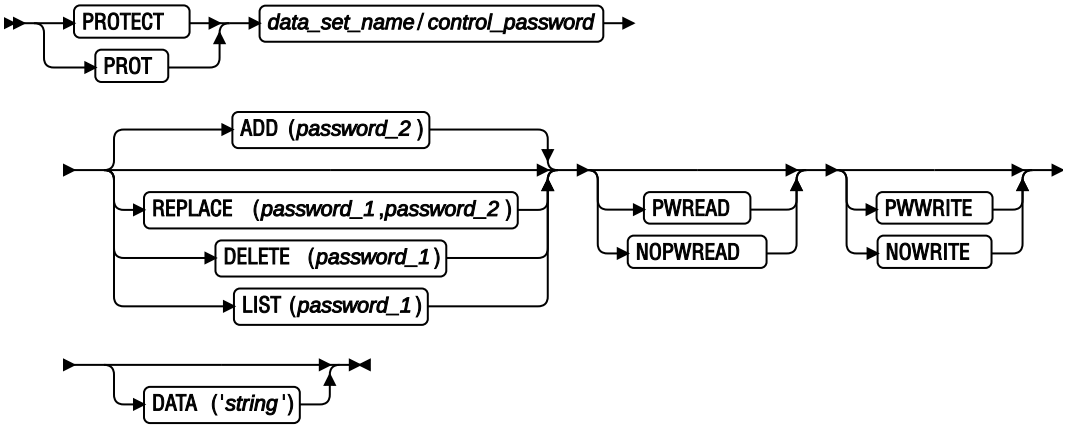
- The passwords that must be specified to gain access to your data.
- The type of access allowed.

Data sets that have been allocated (either during a LOGON procedure or by the ALLOCATE command) cannot be protected by specifying the PROTECT command. To password protect an allocated data set, you need to deallocate it first using the FREE command and then protect it using the PROTECT command.

Note that the PROTECT command does not support dynamic unit control blocks (dynamic UCBs). If the device that holds the data set to be protected has been dynamically reconfigured in your system, you will receive a message explaining that the required volume is not mounted. Instead of using the PROTECT command to control data set access the use of RACF should be considered. For more information about RACF protection, see *z/OS DFSMSdfp Advanced Services*.

The data set password protection that the PROTECT command provides is much weaker than the protection provided by RACF. Many of the reasons for that are stated in the data set password section in *z/OS DFSMSdfp Advanced Services*.

PROTECT command syntax



PROTECT command operands

data_set_name
specifies the name of the data set you want to protect. If the data set is not cataloged, you must specify the fully-qualified name. For example:

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

control_password
Required on all operands except the LIST operand. It provides the control for authorized personnel to alter the password structure on the PROTECT command. See “Password data set” on page 238 for additional information.

ADD | REPLACE | DELETE | LIST

ADD(*password_2*)
specifies a new password is to be required for access to the named data set. ADD is the default.

If the data set exists and is not already protected by a password, its security counter is set and the assigned password is flagged as the control password for the data set. The security counter is not affected when additional passwords are entered.

REPLACE(*password_1*, *password_2*)
specifies that you want to replace an existing password, access type, or optional security information. The first value (password_1) is the existing password; the second value (password_2) is the new password.

DELETE(*password_1*)
specifies that you want to delete an existing password, access type, or optional security information.

If the entry being removed is the control password (see the discussion following these operand descriptions), all other entries for the data set are also removed.

LIST(*password_1*)
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.

password_1
specifies the existing password that you want to replace, delete, or have its security information listed.

password_2
specifies the new password that you want to add or to replace an existing password.

PWREAD | NOPWREAD

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

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

PWWRITE | NOWRITE

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

NOWRITE
specifies the data set cannot be written to.

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).

Passwords

You can assign one or more passwords to a data set. When assigned, the password for a data set must be specified to access the data set. A password consists of 1 to 8 alphanumeric 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: 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 defaults 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.

Password data set

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

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

Each password is stored as part of an entry in the password data set. The first entry in the password data set for each protected data set is called the *control entry*. The password from the control entry must be specified for each access of the data set by using the PROTECT command. However, 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 prompts you for it. If your terminal is equipped with the print-inhibit feature, the system disengages the printing mechanism at your terminal while you enter the password in response. However, the print-inhibit feature is not used if the prompting is for a new password.

PROTECT command return codes

Table 40 on page 238 lists the return codes of PROTECT command.

Table 40: PROTECT command return codes

Return codes	Meaning
0	Processing successful.
12	Processing unsuccessful. An error message has been issued.

PROTECT command examples

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:

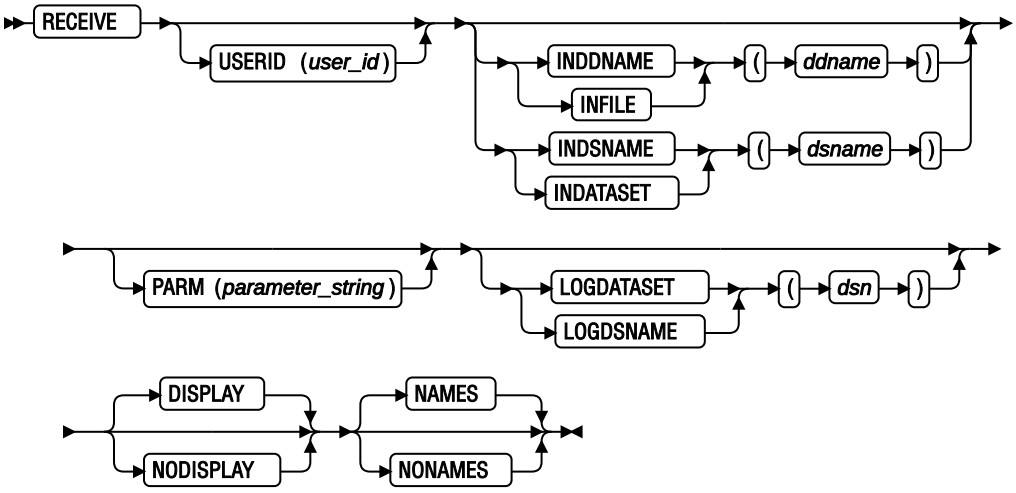
- The name of the data set: ROBID.PROJCTN.LOAD
- The new type of access: NOPWREAD PWWRITE
- The existing password: DDAY6/6
- The control password: EEYORE
- The logon id was: ROBID

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

RECEIVE command

Use the RECEIVE command to retrieve transmitted files and to restore them to their original format.

RECEIVE command syntax



RECEIVE command operands

USERID(*user_id*)
allows you to receive data for a user ID other than your own. The USERID operand is limited to users with OPERATOR authority and to those who are authorized through the RECEIVE initialization exit (INMRZ01). The user ID might exist in SYS1.UADS at the target node or might be a non-existent user ID.

INDDNAME(*ddname*) | INFILE(*ddname*)
specifies the use of a preallocated file as the input data set to receive the transmitted data. Define the data set with RECFM=F, FB, V, VB, or U. For F and FB, LRECL=80. The remaining DCB attributes are installation dependent.

Specify the data set as either sequential or partitioned, but it must be the same as that specified for OUTDDNAME or OUTFILE of the TRANSMIT command. INDDNAME and INFILE are primarily intended for system programmer use.

INDSNAME(*dsname*) | INDATASET(*dsname*)
specifies the use of a sequential data set as the input data set to receive the transmitted data. Define the data set with RECFM=F, FB, V, VB, or U. For F and FB, LRECL=80. The remaining DCB attributes are installation dependent.

If you specify INDATASET or INDSNAME with RECEIVE, the transmitted data is not logged and no acknowledgment is sent to the originator. If you do not specify INDATASET, the transmitted data is logged into the log entry and an acknowledgment is sent to the originator.

Use INDSNAME and INDATASET in combination with OUTDSNAME and OUTDATASET operands of the TRANSMIT command. INDSNAME and INDATASET are primarily intended for system programmer use.

PARM(*parameter_string*)
You can be instructed by your installation to use this operand to specify installation dependent data.

LOGDATASET(*dsname*) | LOGDSNAME(*dsname*)
specifies an alternate name of a sequential data set used to log the transmitted data. RECEIVE checks if the data set, specified by the LOGDATASET/LOGDSNAME operand, is a sequential data set. However, RECEIVE does not check whether the data set attributes are RECFM=VB, LRECL=255, and BLKSIZE=3120. If the data set does not exist, the system creates it.

If you specify NONAMES with LOGDATASET or LOGDSNAME, the system does not search the NAMES data set.

DISPLAY | NODISPLAY

DISPLAY
specifies that the transmitted data or message is to be displayed at the terminal. The system normally displays the data or messages that are transmitted according toone of the following operands of the TRANSMIT command:

- MSGDATASET or MSGDSNAME
- MSGDDNAME or MSGFILE
- MESSAGE or MSG
- TERMINAL

The system places the message or name of the transmitted data set in the log data set. DISPLAY is the default.

NODISPLAY
specifies that the transmitted data or message is not to be displayed at the terminal. The system normally displays the data or messages that are transmitted according to one of the following operands of the TRANSMIT command:

- MSGDATASET or MSGDSNAME
- MSGDDNAME or MSGFILE
- MESSAGE or MSG
- TERMINAL

The system places the data or message in the log data set.

NAMES | NONAMES

NAMES
specifies that RECEIVE search and resolve the NAMES data set for a matching node and user ID of the user who transmitted the data or message. If the nickname and name of the user are found, RECEIVE places the nickname, name, user ID, and node into the log data set.

If the nickname and name are not found, RECEIVE places only the user ID and node into the log data set. All other NAMES data set processing remains the same. For more information about the NAMES data set, see “NAMES data set function” on page 342 under the TRANSMIT command. NAMES is the default.

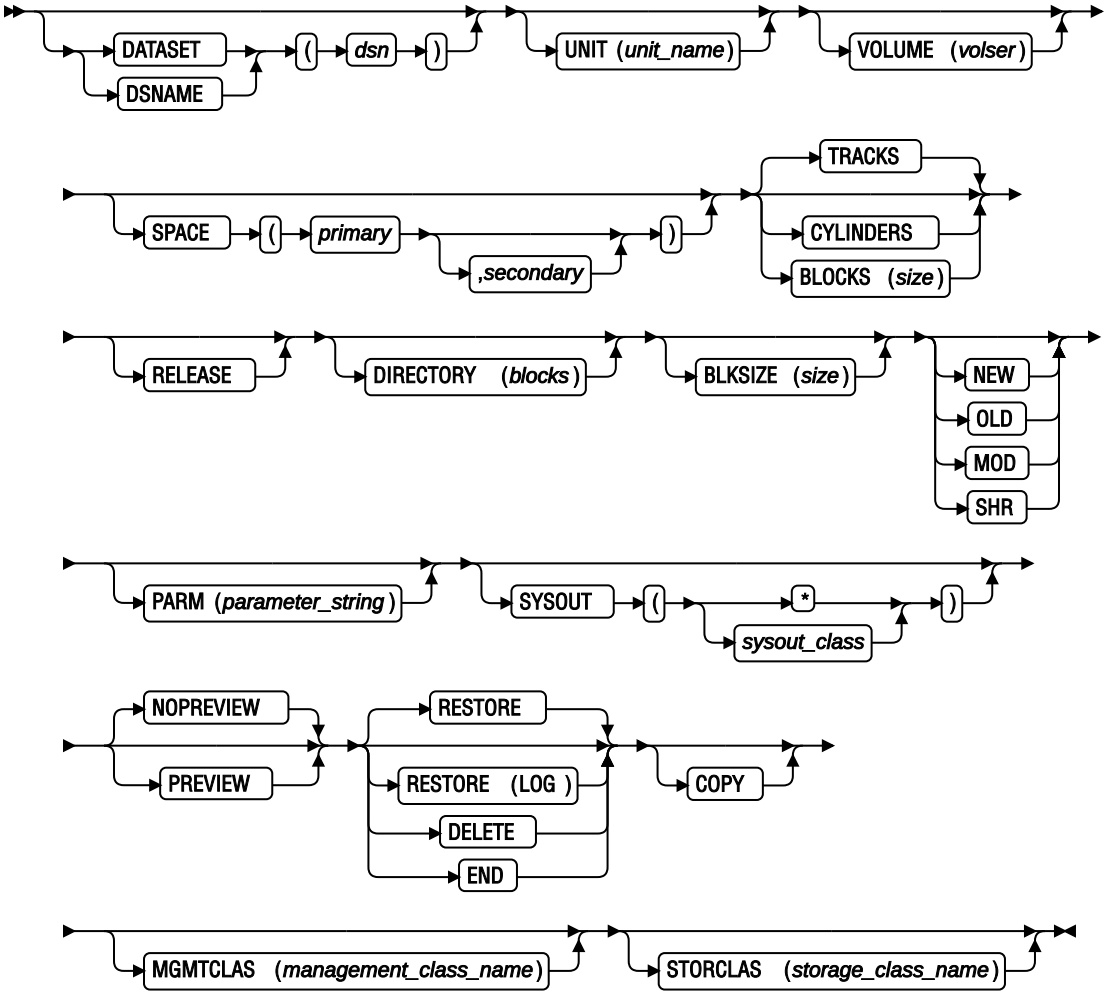
NONAMES
specifies that the nickname and name of the user who transmitted the data or message are not to be resolved. RECEIVE places only the node and user ID in the log data set. All other NAMES data set processing remain the same. For more information about the NAMES data set, see “NAMES data set function” on page 342 under the TRANSMIT command.

If you specify NONAMES with LOGDATASET or LOGDSNAME, the system does not process the NAMES data set.

RECEIVE command prompt parameters

After describing each file, the RECEIVE command *prompts* for overriding parameters. These parameters are all optional and control the restoring of the data set. Parameters not specified are allowed to default or are taken from information transmitted with the data. The optional parameters are shown below.

RECEIVE command prompt parameter syntax



RECEIVE command prompt parameters

Default values for other keywords are specified with the keyword below.

DATASET(*dsname*)|DSNAME(*dsname*)
specifies the name of the data set to be used to contain the received data set. If it does not exist already, the system creates it.

If DATASET and DSNAME are omitted, then RECEIVE uses the name of the transmitted data set, with the high-level qualifier changed to the user ID of the receiving user. If this data set already exists, is a sequential data set, and disposition (SHR/MOD/OLD/NEW) was not specified, RECEIVE prompts you for permission to overwrite the data set. If the data set is partitioned, you are prompted to replace duplicate members.

UNIT(*unit_name*)
specifies a unit name for a new output data set. The default value for UNIT is your normal TSO/E unit name.

VOLUME(*volser*)
specifies a specific volume serial number for a new output data set. The default value for VOLUME is no value, allowing the system to select a volume from those defined by your unit name specified on the UNIT keyword.

SPACE(*primary*,*secondary*)
specifies primary and secondary space for the received data set. The default value for SPACE is a primary size equal to the size of the incoming data and a secondary size of approximately 25 percent

of the primary. If the disposition MOD is used, and the data set is not yet allocated, the system defaults are used to obtain the SPACE parameter defaults.

TRACKS
specifies space to be allocated in tracks. TRACKS is the default when SPACE is specified.

CYLINDERS
specifies space to be allocated in cylinders.

BLOCKS(*size*)
specifies space to be allocated in blocks of the specified size. BLOCKS is the default when SPACE is not specified.

RELEASE
specifies unused space to be released when the receive operation is complete.

DIRECTORY(*blocks*)
specifies an override for the number of directory blocks in a partitioned data set. The default value for DIRECTORY is the number of directory blocks required for the received members.

If a sequential data set is being received into a new PDS by specifying DA(X(MEM)) and DIRECTORY is not specified, the default value for directory blocks is 27.

BLKSIZE(*size*)
specifies a value for the block size of the output data set. This value is used, if it does not conflict with the received data set parameters or device characteristics. BLKSIZE is ignored if specified in response to prompting message INMR907A Enter COPY parameters.

NEW | OLD | MOD | SHR
specifies the data set disposition. If you do not specify one of the disposition keywords and the SPACE value is not present, RECEIVE first tries disposition OLD and attempts to allocate an existing data set. If this fails, disposition NEW is used, space values are added, and another attempt is made at allocation.

PARM(*parameter string*)
Your installation may instruct you to use this operand to specify installation dependent data.

SYSOUT(*sysout_class* | *)
specifies a SYSOUT class to be used for messages from utility programs the RECEIVE command invokes (such as IEBCOPY). If * is specified, these messages are directed to the terminal. The default for SYSOUT is normally *, but this might be changed by the installation.

PREVIEW | NOPREVIEW

PREVIEW
specifies the received data should be displayed at the terminal as it is stored. This is generally appropriate only for sequential data sets because what is displayed is the result of the first pass at restoring the data. For partitioned data sets, the IEBCOPY unloaded format is displayed.

NOPREVIEW
specifies no previewing is to be done. NOPREVIEW is the default.

RESTORE | RESTORE(LOG) | DELETE | END

RESTORE
specifies the transmitted data should be restored to its original format. RESTORE is the default.

RESTORE(LOG)
specifies the transmitted data should be restored to its original format and written to the appropriate log. It is also previewed to the terminal, but it is not written to another data set. You cannot specify RESTORE(LOG) with the DATASET or DSNAME operand. You need to use RESTORE(LOG) primarily to RECEIVE a message and log the message text in the log entry.

DELETE
specifies the file be deleted without restoring it.

END
specifies the RECEIVE command terminate immediately, leaving the current data set on the spool to be reprocessed at a later time.

COPY
specifies not to restore the transmitted data to its original format, but copy it 'as is'. At a later time you can specify RECEIVE INDATASET to restore the data. COPY allows you to examine the data in its transmitted form so that you can debug problems when RECEIVE cannot process the transmitted data. It is primarily intended for system programmer use.

MGMTCLAS(*management_class_name*)
With Storage Management Subsystem (meaning Storage Management Subsystem is installed and is active), specifies the name, 1 to 8 characters, of the management class for a new data set. When possible, do not specify MGMTCLAS. Instead, use the default your storage administrator provides through the automatic class selection (ACS) routines.

After the data set is allocated, attributes in the management class control the following:

- The migration of the data set, which includes migration from primary storage to Data Facility Storage Management Subsystem Hierarchical Storage Manager (DFSMShsm) owned storage to archival storage.
- The backup of the data set, which includes frequency of backup, number of versions, and retention criteria for backup versions.

Note: Without Storage Management Subsystem, the system syntax checks and then ignores the MGMTCLAS operand.

STORCLAS(*storage_class_name*)
with Storage Management Subsystem, specifies the name, 1 to 8 characters, of the storage class. When possible, do not specify STORCLAS. Instead, use the default your storage administrator provides through the automatic class selection (ACS) routines.

The storage class replaces the storage attributes that are specified on the UNIT and VOLUME operand for non-Storage Management Subsystem managed data sets.

A "Storage Management Subsystem-managed data set" is defined as a data set that has a storage class assigned. A storage class is assigned when you specify STORCLAS or an installation-written ACS routine selects a storage class for the new data set.

Note: Without Storage Management Subsystem, the system syntax checks and then ignores the STORCLAS operand.

RECEIVE command return codes

Table 41 on page 244 lists all the return codes of RECEIVE command.

Table 41: RECEIVE command return codes

Return codes	Meaning
0	Processing successful.
4	Processing successful, but a warning message has been issued.
8	Processing incomplete. Some function failed.
12	Processing ends, but is not successful.
16	Processing abnormally terminates.

Receiving data

The RECEIVE command picks the first file that has been transmitted to you, displays descriptive information about the file, and prompts you for information to control the restore operation. You can choose to accept the default data set name (the original data set name with the high-level qualifier changed to the receiving user's TSO/E prefix) and space information or you can override any of these defaults. RECEIVE creates the data set if it does not exist. You can specify a disposition (OLD, SHR, MOD,

or NEW) to force a particular mode of operation. If the data set is successfully restored, RECEIVE continues with the next file. If requested by the sender, RECEIVE generates a notification of receipt and transmits it back to the sender. This return message contains routing and origin information, the name of the data set transmitted, the original transmission sequence number, and an indication of whether the receive was successful. If an error occurred, the message number of the error is included.

You can also use RECEIVE to retrieve Office Vision notes. However, an acknowledgment is not transmitted to the sender of the Office Vision note. Receipt notification is the default for any addressee entered individually on the TRANSMIT command, but not for addressees derived from distribution lists. If you want to be notified for addressees on distribution lists, you must specify :NOTIFY on the distribution list in the control data set or specify NOTIFY(ALL) on the TRANSMIT command.

You can use the RECEIVE command to receive network data (data that was not sent by the TRANSMIT command). The default LRECL for network data is 251 bytes. If you need to receive network data with an LRECL greater than 251 bytes, you must use a data set with an LRECL greater than 251 bytes.

Data set organization

Generally, RECEIVE cannot reformat data sets. The data set into which received data is to be written must have the same record format as the original data set. The record length must be compatible. That is, equal for fixed-length records and equal or longer for variable-length records. The block size of the received data set can be any value that is compatible with the record length and record format. If a mismatch is found in record length, block size, or record format, RECEIVE terminates with appropriate error messages and return codes.

You can receive sequential or partitioned data sets with record formats of F, FS, FB, FBS, V, VB, and U. The largest fixed-length record data set TSO/E can receive from VM is 32,760. Data sets with machine and ASA print-control characters are also supported. RECEIVE does not support data sets with keys, ISAM data sets, VSAM data sets, or data sets with user labels.

Receiving PDSE data sets

RECEIVE supports PDSE Data Libraries and PDSE Program Libraries. Depending on the user's specifications and on the level of MVS/DFP or DFSMS, conversion between PDSEs and PDSs, and PDSs and PDSEs, are performed implicitly. Program Objects which are executable programs stored as members of a PDSE Program Library, are subject to some restrictions when conversions to other data set types are attempted. When executing the TRANSMIT and RECEIVE commands, these restrictions prevent invalid Program Objects from being accepted.

When a PDS or PDSE is sent with the TRANSMIT command, the IEBCOPY utility first copies the data set to an intermediate sequential file. Control information is added and the data set is sent. When it is received, the control information is used to determine the characteristics of the original data set so that a new data set can be allocated with the proper size and of the same type. IEBCOPY is then used to reload the PDS or PDSE. Since there are restrictions when receiving Program Objects, the receiving terminal user gets assistance by means of a message about the data set characteristics of the incoming file. The user is then asked to enter the restore parameters.

Table 42 on page 245 shows all combinations of source and target data sets and how they are handled by TRANSMIT and RECEIVE.

Table 42: Combinations of source and target data sets.

	TARGET						
	Output data set	**Output data set**	**Output data set**	**Output data set**	**No output data set specified**		
SOURCE	Sequential File	PDS	Data Library	Program Library	PDSE is fully supported	Only Data Library is supported	PDSE is not supported

Table 42: Combinations of source and target data sets. (continued)							
	TARGET						
	Output data set	**Output data set**	**Output data set**	**Output data set**	**No output data set specified**		
Sequential File	Sequential File	PDS member	Data Library member	INMR155I	Sequential File	Sequential File	Sequential File
PDS	Sequential File *	PDS	Data Library	INMR156I	PDS	PDS	PDS
Data Library	Sequential File *	PDS	Data Library	INMR157I	Data Library	Data Library	PDS
Program Library	INMR158I	INMR158I	INMR158I	Program Library	Program Library	INMR159I	INMR159I
* PDS member or Data Library member was transmitted with the SEQ option.							
INMR15xI is the error message number resulting from the user request.							

Receiving protected data sets

RECEIVE warns you if you are receiving a data set that was RACF or PASSWORD protected. It takes no further action to protect newly restored data. If you are using the automatic data set protection feature of RACF or a RACF generic profile, the data set is protected. Otherwise, use the PROTECT command or the RACF ADDSD command to protect the data.

Receiving enciphered data

If RECEIVE detects that TRANSMIT enciphered the incoming file, it automatically attempts to decipher the data. To do this, it prompts you for decipher options and then passes these to the Access Method Services REPRO command. See “Data encryption function of TRANSMIT and RECEIVE” on page 340.

The RECEIVE command logs transmissions. See “Logging function of TRANSMIT and RECEIVE” on page 341.

Receiving data sets and messages with security labels

If your installation uses security labels and security options, any data sets or messages transmitted to you have a security label associated with them. In order for you to receive the data, you must be logged on at a security label equal to or greater than the security label with which the data was transmitted.

Some considerations for receiving data sets and messages with security labels are:

- You can only receive data sets and messages you are authorized to receive based on the security label you are logged on with.
- To receive data sets and messages with a greater security label, you can log on with a greater security label if your TSO/E user ID is authorized to do so. Then you can use the RECEIVE command to view the messages and data sets.
- If you cannot log on with a security label that allows you to receive the data set or message, the system deletes the data, unless your installation uses a JES installation exit to take some other action.
- You do not receive a notice that you have data sets or messages to receive if they were transmitted with a security label that is greater than the security label with which you are logged on.

RECEIVE command examples

In the following examples, the transmitting user is assumed to have user ID USER1 on node NODEA and the receiving user is assumed to have user ID USER2 on node NODEB. The sending user has a NAMES data set as follows:

```
* Control section
:altctl.DEPT.TRANSMIT.CNTL
:prolog.Greetings from John Doe.
:prolog.
:epilog.
:epilog.Yours,:epilog.John Doe :epilog.NODEA.USER1
*
* Nicknames section.
*
:nick.alamo   :list.Jim Davy      :logname.alamo   :notify.
:nick.addrchg :list.joe davy jim  :nolog           :nonotify
:nick.Joe     :node.nodeb  :userid.user2    :name.Joe Doe
:nick.Me      :node.nodea  :userid.user1    :name.me
:nick.Davy    :node.alamo  :userid.CROCKETT :name.Davy Crockett
:nick.Jim     :node.ALAMO  :userid.Bowie    :name.Jim Bowie
```

In the examples involving the RECEIVE command, data entered by the user appears in lowercase and data displayed by the system is in uppercase.

Example 1

Transmit a copy of the ‘SYS1.PARMLIB’ data set to Joe, identifying Joe by his node and user ID.

```
transmit nodeb.user2 da('sys1.parmlib')
```

Example 2

Joe receives the copy of ‘SYS1.PARMLIB’ transmitted above.

```
receive
Dataset SYS1.PARMLIB from USER1 on NODEA
Enter restore parameters or 'DELETE' or 'END' +
<null line>
Restore successful to dataset 'USER2.PARMLIB'
-----------------------------------------------
No more files remain for the RECEIVE command to process.
```

In the preceding example, Joe has issued the RECEIVE command, seen the identification of what arrived, and chosen to accept the default data set name for the arriving file. The default name is the original data set name with the high-level qualifier replaced by his user ID.

Example 3

Transmit two members of ‘SYS1.PARMLIB’ to Joe, and add a message identifying what was sent. Joe is identified by his NICKNAME, leaving it to TRANSMIT to convert it into node and user ID by the nicknames section of the NAMES data set.

```
transmit joe da('sys1.parmlib') mem(ieasys00) line
ENTER MESSAGE FOR NODEB.USER2
Joe,
   These are the parmlib members you asked me to send you.
They are in fact the ones we are running today.
Yours,
John Doe
<null line>
```

The message text in this example was entered in line mode which would be unusual for a user on a 3270 terminal, but which is easier to show in an example.

Example 4

Joe begins the receive process for the members transmitted in Example 3 and ends the receive without actually restoring the data onto the receiving system, because Joe does not know where he wants to store the data.

```
receive
Dataset SYS1.PARMLIB  from USER1 on NODEA
Member: IEASYS00
Greetings from John Doe.
Joe,
   These are the parmlib members you asked me to send you.
They are in fact the ones we are running today.
Yours,
John Doe
NODEA.USER1
Enter restore parameters or 'DELETE' or 'END' +
end
```

In the preceding example, notice that the PROLOG and EPILOG lines have been appended to the message entered by the sender. In an actual RECEIVE operation, the original message text would appear in both uppercase and lowercase just as the sender had entered it (assuming the receiver's terminal supports lowercase.)

Example 5

Joe receives the 'SYS1.PARMLIB' members transmitted in Example 3. Specify space parameters for the data set that will be built by RECEIVE to leave space for later additions.

```
receive
Dataset SYS1.PARMLIB from USER1 on NODEA
Members: IEASYS00
Greetings from John Doe.
Joe,
   These are the parmlib members you asked me to send you.
They are in fact the ones we are running today.
Yours, John Doe
NODEA.USER1
Enter restore parameters or 'DELETE' or 'END' +
da('nodea.parmlib') space(1) cyl dir(10)
Restore successful to dataset 'NODEA.PARMLIB'
------------------------------------------------
No more files remain for the RECEIVE command to process.
```

The received member IEASYS00 is saved in the output data set with their member names unchanged.

Example 6

Send a message to a user on another system. For more information about the TRANSMIT command, see the "TRANSMIT command" on page 334.

```
transmit davy
```

The system displays the following screen for input:

```
                    DATA FOR ALAMO.CROCKETT
0001   Davy,
0002     Did you check the report I gave you last week?
0003   Joe
0004
0005
:
```

Press PF3 to send the message.

In this example, the target user is identified by his nickname and no data set is specified, causing the terminal to be used as an input source. You can type your data, scroll using program function (PF) keys PF7 or PF19 and PF8 or PF20, and exit using PF3 or PF15, or cancel using the PA1 key.

Example 7

Send a member of a partitioned data set as a message. In this example, the member MEETINGS of the partitioned data set MEMO.TEXT is sent as a message to JOE.

```
transmit nodeb.joe msgds(memo.text(meetings))
 INMX000I 0 message and 7 data records sent as 5 records to NODEB.JOE
 INMX001I Transmission occurred on 07/27/87 at 09:00:35.
READY
```

JOE receives the message in his data set MY.LOG, instead of the default log data set, LOG.MISC:

```
receive logds(my.log)
 INMR901I Dataset ** MESSAGE ** from MIKE on NODED
 THIS IS A SCHEDULE OF STATUS MEETINGS FROM AUGUST THROUGH NOVEMBER:

 AUGUST      MONDAYS AT 9:00 A.M. IN MY OFFICE
 SEPTEMBER   TUESDAYS AT 10:00 A.M. IN YOUR OFFICE
 OCTOBER     WEDNESDAYS AT 10:00 A.M. IN JACK'S OFFICE
 NOVEMBER    MONDAYS AT 2:00 P.M. IN JILL'S OFFICE
```

RENAME command

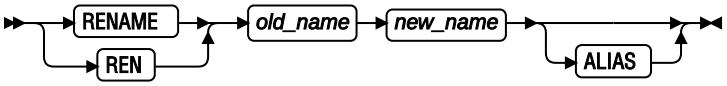
Use the RENAME command to:

- Change the name of a single-volume, non-VSAM cataloged, non-SMS managed data set.
- Change the name of a single or multi-volume, non-VSAM cataloged, SMS managed data set.
- Change the name of a member of a partitioned data set
- Create an alias for a member of a partitioned data set.

The access method services ALTER command changes the name of VSAM data sets and is described in *z/OS DFSMS Access Method Services Commands*.

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 command syntax



RENAME command operands

old_name
specifies the name that you want to change. The name that you specify can 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 can supply only the member name and omit all other levels of qualification.

ALIAS
specifies 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 changes all data set names that match the old name except for the qualifier corresponding to the asterisk's position.

Note: Do not use the RENAME command to create an alias for a linkage editor created load module.

RENAME command return codes

Table 43 on page 250 lists the return codes of RENAME command.

Table 43: RENAME command return codes

Return codes	Meaning
0	Processing successful.
12	Processing unsuccessful. An error message has been issued.

RENAME command examples

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 can specify either: 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

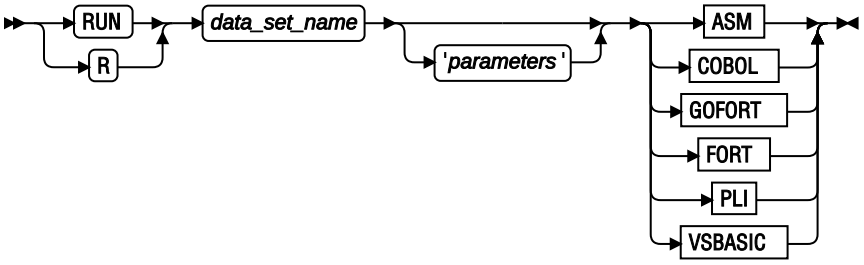
Use the RUN command to compile, load, and execute the source statements in a data set. The RUN command is designed specifically for use with certain licensed programs. It selects and invokes the particular licensed program needed to process the source statements in the data set that you specify. Table 44 on page 250 shows which licensed program is selected to process each type of source statement.

Table 44: Source statement/licensed program relationship

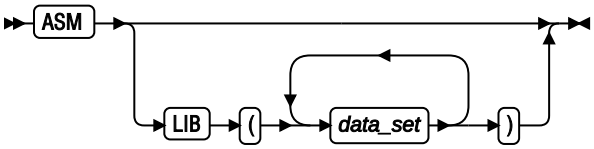
Source	Licensed program
Assembler	Assembler (F)
COBOL	OS/VS COBOL Release 2.4
FORTRAN	FORTRAN IV (G1)
PLI	PL/I Checkout Compiler or PL/I Optimizing Compiler
VSBASIC	VSBASIC

The RUN command and the RUN subcommand of EDIT perform the same basic function.

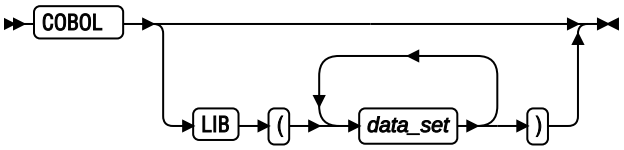
RUN command syntax



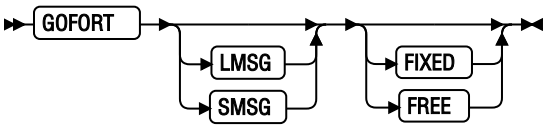
ASM



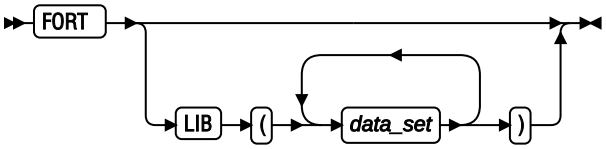
COBOL



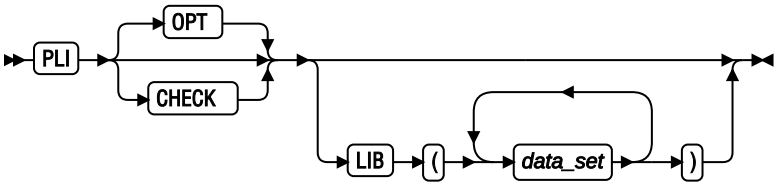
GOFORT



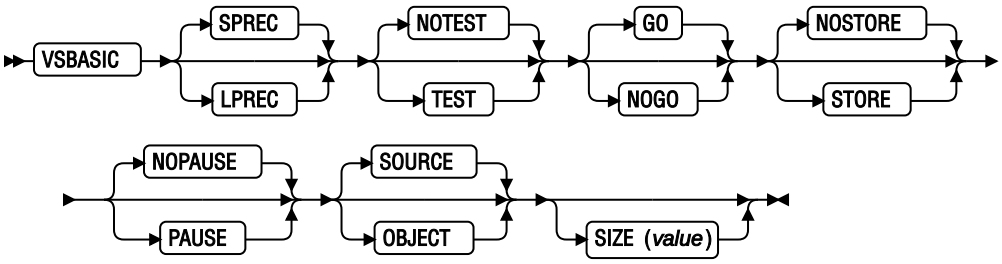
FORT



PLI



VSBASIC



RUN command operands

data_set_name 'parameters'
specifies the name of the data set containing the source program. A string of up to 100 characters can be passed to the program by the parameters operand (valid only for data sets which accept parameters).

ASM
specifies that the Assembler (F) is to be called to process source programs.

If the rightmost qualifier of the data set name is ASM, this operand is not required.

LIB(*data_set*)
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 the OS/VS COBOL licensed program is to be invoked to process the source program. If the rightmost qualifier of the data set name is COBOL, this operand is not required.

GOFORT
specifies the Code and Go FORTRAN licensed program is to be invoked to process the source program. If the rightmost qualifier of the data set name is GOFORT, this operand is not required.

LMSG | SMSG

LMSG
specifies long form diagnostic messages are to be provided.

SMSG
specifies short form diagnostic messages are to be provided.

FIXED | FREE

FIXED
specifies statements adhere to the standard FORTRAN column requirements and are 80 bytes long.

FREE
specifies statements are of variable lengths and do not conform to set column requirements.

FORT
specifies the TSO FORTRAN Prompter and the FORTRAN IV (G1) licensed programs are to be invoked to process the source program. If the rightmost qualifier of the data set name is FORT, this operand is not required.

PLI
specifies either the PL/I Optimizer compiler or the PL/I Checkout compiler is to be invoked to process the source program. If the rightmost qualifier of the data set name is PLI, this operand is not required.

CHECK | OPT

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. If both CHECK and OPT are omitted, OPT is the default value.

VSBASIC
specifies the VSBASIC licensed program is to be invoked to process the source program.

LPREC | SPREC

LPREC
specifies long precision arithmetic calculations are required by the program.

SPREC
specifies short precision arithmetic calculations are adequate for the program. SPREC is the default value.

TEST | NOTEST

TEST
specifies testing of the program is to be performed.

NOTEST
specifies the TEST function is not to be performed. NOTEST is the default value.

GO | NOGO

GO
specifies the program is to receive control after compilation. GO is the default value.

NOGO
specifies the program is not to receive control after compilation.

STORE | NOSTORE

STORE
specifies the compiler is to store an object program.

NOSTORE
specifies the compiler is not to store an object program. NOSTORE is the default value.

PAUSE | NOPAUSE

PAUSE
specifies the compiler is to prompt to the terminal between program chains.

NOPAUSE
specifies no prompting between program chains. NOPAUSE is the default value.

SOURCE | OBJECT

SOURCE
specifies the new source code is to be compiled. SOURCE is the default value.

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

SIZE(*value*)
specifies the number of 1000-byte blocks of user area where value is an integer of one to three digits.

To fully qualify the data set name, the RUN command adds the suffix ASM, COBOL, GOFORT, FORT, PLI, or VSBASIC to the data set name, unless the data set name is already suffixed by the specified operand.

Determining compiler type

The system uses two sources of information to determine which compiler is to be used. The first source of information is the optional operand (ASM, COBOL, GOFORT, FORT, PLI, or VSBASIC) that you can 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. If the system cannot determine the compiler type from the descriptive qualifier, you are prompted for it unless PROFILE NOPROMPT is in effect.

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 or binder to locate load modules referred to by the module being processed for resolution of external references.

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

RUN command return codes

Table 45 on page 254 lists the return codes of RUN command.

Table 45: RUN command return codes

Return codes	Meaning
0	Processing successful.
12	Processing unsuccessful. An error message has been issued.

RUN command examples

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
```

SEND command

Use the SEND command or the SEND subcommand of EDIT to send a message to anyone of the following destinations:

- One or more users
- An operator specified by route code
- An operator console specified by name

SEND can be used to send a message from one user to another user in the same JESPLEX.

In order for the recipient to receive and display the message, the recipient's profile must include the INTERCOM operand. To change the profile, use the PROFILE command.

By default, when you issue the SEND command with the NOW operand, the message is displayed on the recipient's screen if he or she is logged on and receiving messages. If the receiver is not logged on or is not receiving, the message is deleted and you receive a message stating why the message was not displayed.

If you issue SEND with the LOGON operand and the recipient is logged on and receiving, the message is also displayed. If the recipient is not logged on when you send the message, the message is stored in the broadcast data set. (If your installation uses individual user logs, SEND stores the message in the user log, truncating trailing blanks. Otherwise, SEND stores the message in the broadcast data set and does not truncate trailing blanks.)

When you issue SEND with the SAVE operand, the message is stored even if the recipient is logged on. SEND stores the message in the broadcast data set unless your installation uses individual user logs, in which case, the message is stored in the recipient's user log.

Installations can use security enhancements to customize how the SEND command works. For example, using RACF, an installation can control which users can send messages to other users. If your installation uses these security features and you send a message to a user that you are not authorized to send messages to, the system cancels your message and displays an informational message on your terminal.

SEND also works differently if your installation uses security labels and security checking. Each time you send a message, the security label you are logged on with is associated with the message. The security label is used to determine if the recipient can view the message.

If you issue SEND NOW or SEND LOGON, the sender's current security label is associated with the message. If the recipient is logged on at a security label that is equal to or greater than the security label of the message, the message is displayed immediately. If SEND NOW was specified and the recipient's security label is less than the message's security label, the message is canceled. If SEND LOGON was specified and the recipient's security label is less than the message's security label, the message is saved in the recipient's user log.

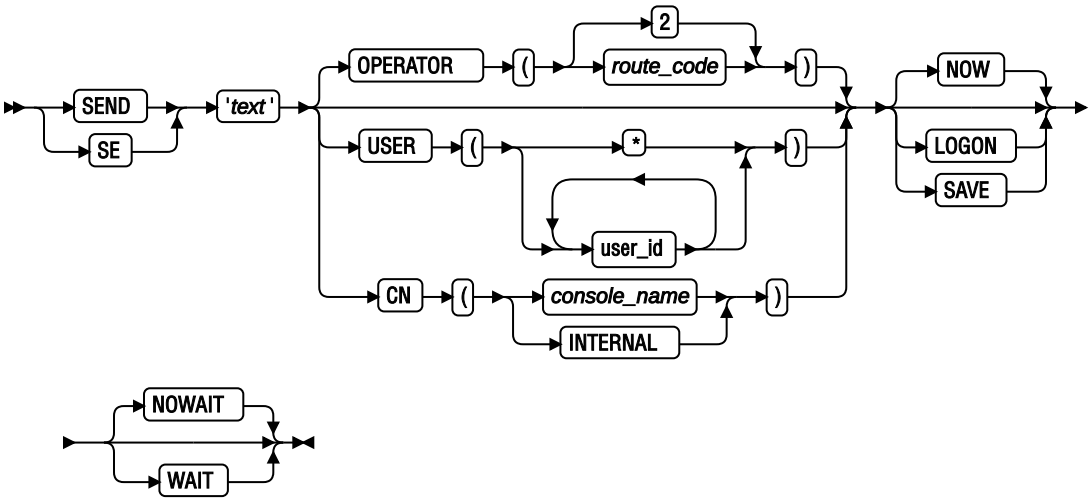
If the message you send is to be saved (when you issue SEND SAVE or SEND LOGON and the recipient is not logged on), the message's security label is saved along with the message in the recipient's user log. The recipient can view messages saved in the user log by issuing the LISTBC command. All messages in the recipient's user log that have a security label that is less than or equal to the security label of the recipient are then displayed. Messages that have a security label higher than the security label the recipient logged on with are not displayed. See "LISTBC command" on page 165 for more information about receiving messages from a user log.

If your installation uses security labels and security checking, messages are stored in user logs. If you try to send a message to another user who does not have a user log and the message is to be saved, the message is not sent. You get a message explaining why the message cannot be saved.

Note: The SEND command can not be used to communicate with users who are logged in under z/OSMF ISPF.

The syntax and function of the SEND subcommand of EDIT is the same as that of the SEND command.

SEND command syntax



SEND command operands

'text'

Specifies the message to be sent. You must enclose the text of the message within apostrophes (single quotation marks). The message cannot 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 apostrophes to get one.

USER(*user_id* | *)

user_id

Specifies the user identification of one or more terminal users who are to receive the message. A maximum of 20 identifications can be used. The message is routed to the system within the sysplex to which the recipient is logged on.

***

Specifies the message is sent to the user ID associated with the issuer of the SEND command. If * is used with a SEND command in a CLIST, the message is sent to the user executing the CLIST. If it

is used with the SEND command at a terminal, * causes the message to be sent to the same terminal.

OPERATOR(2 | *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; that is, the message goes to the operator. If both USER (identification) and OPERATOR are omitted, OPERATOR is the default. The integer corresponds to routing codes for the write-to-operator (WTO) macro.

If you send a message with a length of greater than 72 characters to OPERATOR or a console using the SEND command, the message is issued as two WTOs.

CN(*console_name*| INTERNAL)
Specifies the message is to be queued to the indicated operator console.

console_name
Is 2 to 8 alphanumeric characters, the first of which must be alphabetic or one of the special characters #, $, or @.

Note: Except for extended MCS consoles, console names are defined by your installation.

INTERNAL
Specifies that the message is to be sent to any active console defined with INTIDS=Y. For additional information about defining consoles with INTIDS=Y, see *z/OS MVS Planning: Operations*.

If you send a message with a length of greater than 72 characters to OPERATOR or a console using the SEND command, the message is issued as two WTOs.

NOW | LOGON | SAVE

NOW
Specifies that you want the message to be sent immediately. If the recipient is not logged on or is not receiving messages, you are notified and the message is deleted. If your installation uses security labels and security checking and the recipient is logged on and is receiving messages, but does not have an appropriate security label to view the message, you are notified and the message is deleted. NOW is the default value.

LOGON
Specifies that you want the message retained in the SYS1.BRODCAST data set or the user log data set, if the recipient is not logged on, is not receiving messages, or cannot receive messages because of SECLABEL checking. If the recipient is using the system and receiving messages, the message is sent immediately. If your installation uses security labels and security checking, and the recipient is logged on, is receiving messages, and has an appropriate security label to view the message, then the message is sent immediately. Otherwise, the message is saved, and the recipient must issue LISTBC or LOGON specifying MAIL to retrieve the message.

Using LOGON, a message can be saved for retrieval by a user on any system where the SYS1.BRODCAST data set or user log data set is properly shared.

Note: When processing messages by using the SYNC, LISTBC, and SEND commands, the SYS1.BRODCAST data set is not used for 8 character user IDs. Any messages that cannot be delivered to the user's screen because the user is not logged on or the keywords on the command are only stored if user logs are enabled by using IKJTSO*xx*.

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

Using SAVE, a message can be saved for retrieval by a user on any system where the SYS1.BRODCAST data set or user log data set is properly shared.

NOWAIT | WAIT

NOWAIT
Specifies that you do not want to wait if system output buffers are not immediately available for all specified logged-on terminals. You are notified of all specified users who did not receive the message. If you specified LOGON, mail is created in the SYS1.BRODCAST data set or the user log data set for the specified users whose terminals are busy or who have not logged on. NOWAIT is the default value.

WAIT
Specifies that you want to wait until system output buffers are available for all specified logged on terminals. This ensures that the message is received by all specified logged on users, but it also means that you might be locked out until all such users have received the message.

SEND command return codes

Table 46 on page 257 lists the return codes of SEND command.

Table 46: SEND command return codes

Return codes	Meaning
0	Processing successful.
12	Processing unsuccessful.

The return codes listed in Table 47 on page 257 are valid only if you have an installation-defined user log data set:

Table 47: SEND command return codes (installation-defined user log data set)

Return codes	Meaning
0	Message was successfully sent for display; all users received it.
4	Message was successfully stored. Either the user is not logged on, or is not logged on with a security label that allows the user to view the message.
8	Message was successfully stored; saved the message.
12	Message was not displayed; user is not logged on.
16	Message was not displayed; user's terminal is busy.
18	Sender not permitted to send messages to one or more specified users.
20	Message was not displayed; user is not accepting messages.
22	Message cannot be viewed by one or more specified users; their security label is lower than the sender's security label.
24	Message was not stored; saving is not allowed.
26	One or more users did not have an individual user log and the message can not be saved in the broadcast data set.
28	Message was not stored; user log unavailable.
32	Message was not sent; user denied access.
36	Message was not sent; SEND is inactive.
40	Message was not sent; no such user ID.

Table 47: SEND command return codes (installation-defined user log data set) (continued)	
Return codes	**Meaning**
44	Message was not sent; command is not authorized.
92	Message was not sent; system error.

SEND command examples

Example 1

Operation: Send a message to the 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 and LYNN6
- The message is important and you want to wait until the recipients have received the message.

```
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 recipient's user identification: BETTY7
- The message: Is your version of the simulator ready?
- If she is not logged on, you want to save the message until she logs on again. There is no rush for her to get it and to respond to it.

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

Example 4

Operation: Send a message to the operator console ‘TAPELIB’.

Known:

- The console name: TAPELIB
- The message: Please mount tape number A021. I need it to run my program.

```
send 'Please mount tape number A021. I need -
it to run my program.'  CN(TAPELIB)
```

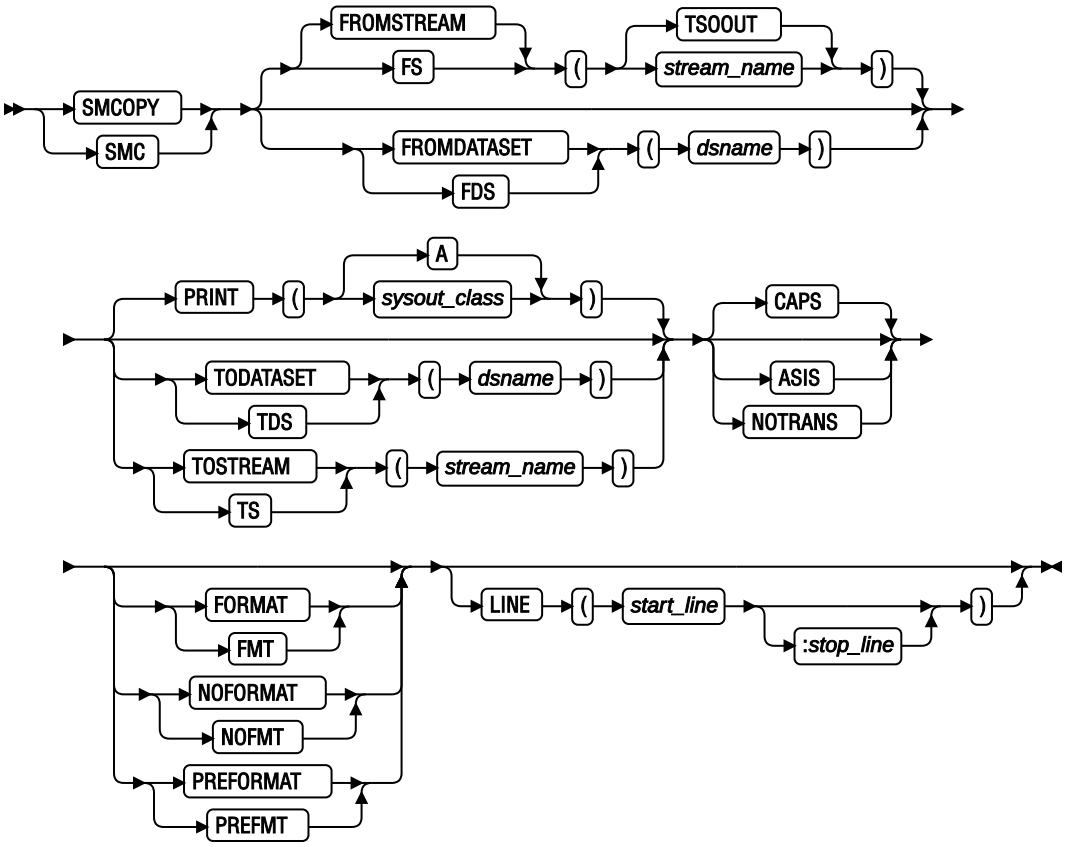
SMCOPY command

Use the SMCOPY command to copy all or part of a stream or data set to another stream or data set (that is, stream to stream, stream to data set, data set to stream, or data set to data set).

Note:

1. When using SMCOPY under ISPF, you must be logged on to Session Manager to copy TSOOUT and TSOIN streams. Also, be certain SESSMGR is set to YES in ISPF. For information about setting SESSMGR to YES, see *z/OS ISPF Planning and Customizing*.
2. If the source and target of the copy request are both data sets, (SYSOUT or QSAM), you do not have to be logged on under the Session Manager to use the SMCOPY command.

SMCOPY command syntax



SMCOPY command operands

FROMDATASET(*dsname*)
specifies the name of the data set that contains the information to be copied. The data set must be a sequential data set or a member of a partitioned data set with either fixed- or variable-length records. The data set must reside on a volume that is mounted or on a device that is on-line.

FROMSTREAM([*stream_name*])
specifies the name of the input stream that contains the information to be copied. If you do not specify FROMDATASET or FROMSTREAM, the default is FROMSTREAM. If you do not specify stream_name, the default is TSOOUT.

PRINT(*sysout_class*)
specifies that the information is to be copied to a SYSOUT data set of the specified SYSOUT class and printed on a system printer. You can print up to 132 characters per line. If PRINT, TODATASET, and TOSTEAM are omitted, then SMCOPY assumes PRINT(A).

TODATASET(*dsname*)
specifies the name of the data set into which the information is to be copied. The data set must be sequential or a member of a partitioned data set with either fixed-or variable-length records. The data set must reside on a volume that is mounted or on a device that is on-line.

If the data set does not exist, the Session Manager allocates a new data set. If the information is being copied from a data set (the FROMDATASET operand is specified), the attributes from this data set are used except for the size which defaults to 5 tracks primary and 5 tracks secondary space. If more space is required for the data set than the default provides, you must preallocate the data set. If the information is being copied from a stream (the FROMSTREAM operand is specified), the new data set is allocated with the following attributes:

RECFM
VB or VBA if FORMAT or PREFORMAT is specified.

LRECL
<256

BLKSIZE
3120

TOSTREAM(*stream_name*)
specifies the name of the output stream for the copy operation.

ASIS | CAPS | NOTRANS

ASIS
specifies that the Session Manager is to leave lowercase letters as lowercase letters and translate the unprintable characters to blanks (X'40').

Use the ASIS operand if the information is to be printed on a printer with a dual-case print train (such as TN or T11).

CAPS
specifies that the Session Manager is to translate lowercase letters to uppercase and translate the unprintable characters to blanks (X'40').

NOTRANS
specifies that no translation is to occur.

FORMAT | NOFORMAT | PREFORMAT

FORMAT
specifies that carriage control characters are to be placed in the copied information. If the information is being placed in a stream, the highlighted lines are highlighted in the stream.

If the information is being copied to a data set, the record format must be FBA or VBA to indicate the presence of ASA control characters. If the data set is new, the Session Manager allocates it with a VBA record format.

FORMAT is ignored if FROMSTREAM is not specified.

NOFORMAT
specifies that no control characters are to be placed in the copied information.

If the information is being copied from a data set, the data set must have a FB or VB record format. If the information is being copied from a stream to a data set, the data set must have a FB or VB record format. If the information is being copied from a data set to a data set, both data sets must have the same format (FB or VB). If the data set that the information is going into is new, the Session Manager allocates it with a VB record format (if it is being copied to a stream) or it is allocated with the same record format as the data set it is coming from (for a data set to data set copy operation).

PREFORMAT
specifies that the source for the copy (stream or data set) already contains carriage control characters. Use this operand when the SNAPSHOT command was previously used to place information in a stream or data set.

If the information is being copied from a data set, the data set must have a FBA or VBA record format. If the information is being copied from a stream to a data set, the data set must have a FBA or VBA record format. If the information is being copied from a data set to a data set, both data sets must have the same format (FBA or VBA). If the data set that the information is going into is new, the Session Manager allocates it with a VBA record format (if it is being copied to a stream) or it is allocated with the same record format as the data set it is coming from (for a data set to data set copy operation).

LINE(*start_line* |*stop_line*)
specifies the range of lines to be copied. The default is the first line of the information and the last line of the information.

If the information is being copied from a stream, you can find specific line numbers by using the QUERY, SMFIND, or FIND.LINE commands. If the information is being copied from a data set, 'LINE' represents records of the data set, not the line numbers within a numbered data set.

SMCOPY command return codes

Table 48 on page 261 lists all the return codes of SMCOPY command.

Table 48: SMCOPY command return codes

Return codes	Meaning
0	Processing successful.
4	Processing successful. Copy operation ended at the end of file or at the end of stream.
8	Processing unsuccessful. The copy was not performed.
12	Processing unsuccessful. Internal error, contact your system programmer.

SMCOPY command examples

Example 1

Copy the TSOOUT stream to the system printer, translating all lowercase letters to uppercase.

```
smcopy
```

Example 2

Copy the member ZLOGON of the data set 'SYS1.CLIST' to the member ZLOGON of the data set TEST.CLIST.

```
smcopy fromdataset('sys1.clist(zlogon)')
       todataset(test.clist(zlogon))
```

Example 3

Copy the data set containing TSO/E commands from the data set SAMPLE and place these commands in the TSOIN stream where they will be executed.

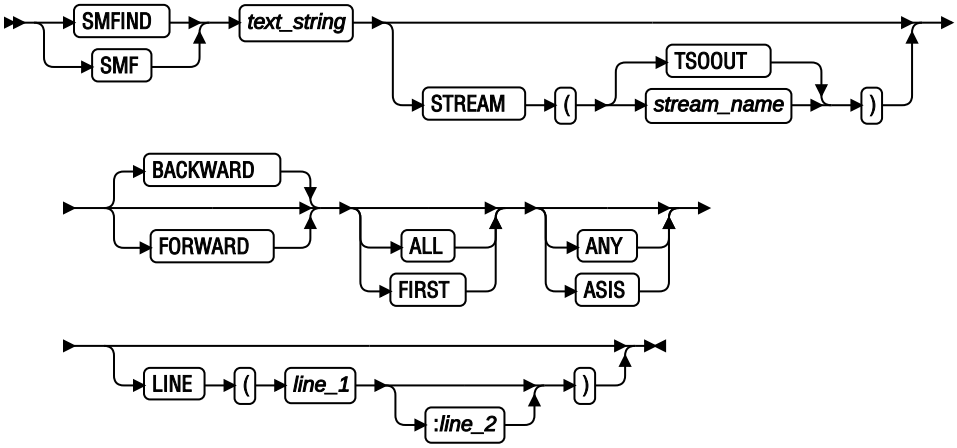
```
smcopy fromdataset('sample.commands.data')
       tostream(tsoin)
```

SMFIND command

Use the SMFIND command to locate a string of characters in a stream. If the text string is found, the Session Manager displays the line number of the text string in the output stream for the TSO/E function (TSOOUT in the default environment) and puts the line number in register 15. If operating from a CLIST, you can access the line number from the CLIST variable '&LASTCC'.

Note: SMFIND command processing assumes that the last line of the output stream is the SMFIND command. Therefore, the SMFIND command does not search the last line of the output stream.

SMFIND command syntax



SMFIND command operands

text_string
specifies the string of characters to be found. The *text_string* can be up to 256 characters in length and must be enclosed in delimiters that are not present in the *text_string*.

STREAM(*stream_name*)
specifies the name of the stream to be searched.

BACKWARD | FORWARD

BACKWARD
specifies that the Session Manager is to search for the *text_string* from the current location backward toward the top of the stream.

FORWARD
specifies that the Session Manager is to search for the *text_string* from the current location forward toward the bottom of the stream.

ALL | FIRST

ALL
specifies that the Session Manager is to find all occurrences of the *text_string*. The line number of each found *text_string* is displayed in the output stream for the TSO/E function. Register 15 (and the CLIST variable &LASTCC) contains the line number of the last occurrence of the *text_string*.

FIRST
specifies that the Session Manager is to find only the first occurrence of the *text_string*. The Session Manager displays the line number of the found *text_string* in the output stream for the TSO/E function. It also places the number in register 15 and the CLIST variable &LASTCC.

ANY | ASIS

ANY
specifies that upper and lowercase differences are to be ignored when finding the *text_string*.

ASIS
specifies that the Session Manager is to find an exact match of the entered *text_string*.

LINE(*line_1*:*line_2*)
specifies the range of lines to be searched.

If only *line_1* is specified, the Session Manager searches from that line to the top or bottom of the stream depending on whether BACKWARD or FORWARD is specified.

If you specify a value for *line_1* or *line_2* that is not in the stream, the Session Manager uses the top or bottom line in the stream.

SMFIND command return codes

Upon completion, SMFIND returns the following:

Table 49 on page 263 lists the return codes of SMFIND command.

Table 49: SMFIND command return codes

Return code	Meaning
0	Return code 0 means one of the following: • The *text_string* was not found. • The specified stream was not found. • The command was incorrectly specified and SMFIND was unable to prompt for correct information.
Other	A positive integer specifying the line number of the found *text_string*. The maximum value is 16,777,216.

SMFIND command examples

Example 1

Find the next occurrence of ‘time’ in the TSOOUT stream.

```
smfind 'time' forward
```

Example 2

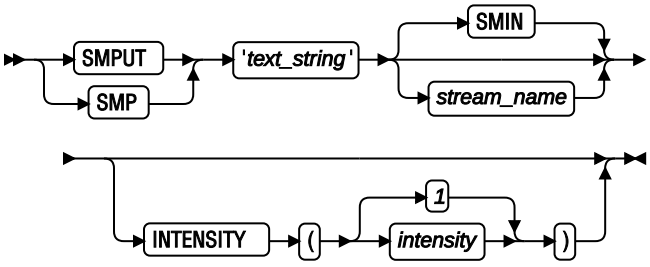
Find the previous occurrence of ‘time’ in the TSOOUT stream.

```
smfind 'time'
```

SMPUT command

Use the SMPUT command to place a text string in a stream. If you place the text string in the TSOIN stream, it is interpreted as a TSO/E command. If you place the text string in the SMIN stream, it is interpreted as a Session Manager command.

SMPUT command syntax



SMPUT command operands

text_string
specifies the string of characters to be placed in the stream. The *text_string* must be enclosed in delimiters that are not in the *text_string*. It can be up to 32768 characters in length, excluding the delimiters. If the *text_string* is being sent to the SMIN stream, it can be up to 512 characters in length.

stream_name
specifies the name of the stream where the *text_string* is to be placed.

INTENSITY(*intensity* | 1)
specifies the brightness at which the information in the stream is to be displayed. The valid values are:

0
The information in the stream is not to be displayed. You can see the line that the information occupies, but the information itself is invisible.

1
The information is to be displayed at normal intensity.

2
The information is to be highlighted.

Note: You must specify *stream_name* if you specify a value for INTENSITY.

SMPUT command return codes

Table 50 on page 264 lists all the return codes of SMPUT command.

Table 50: SMPUT command return codes

Return code	Meaning
0	Processing successful. **Note:** If the *text_string* contained Session Manager or TSO/E commands, the zero return code does not indicate successful execution of those commands.
4	Syntax error.
8	The stream was not found.
12	Processing unsuccessful.

SMPUT command examples

Example 1

Place the TSO/E TIME command highlighted in the TSOIN stream.

```
smput 'time' tsoin intensity(2)
```

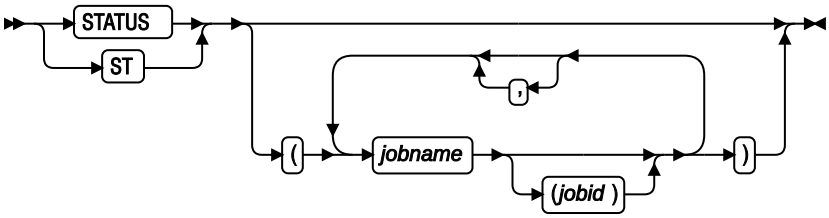
STATUS command

Use the STATUS command to have the status of batch jobs displayed at your terminal. You can obtain the status of all batch jobs, several specific batch jobs, or a single batch job. The information you receive for each job tells you whether it is awaiting execution, is currently executing, or has completed execution but is still on an output queue. It also indicates whether the job is in hold status. An attention interrupt during the processing of STATUS results in termination of the command, but not the job.

STATUS is a foreground-initiated-background (FIB) command. You must be authorized by installation management to use STATUS. This command is generally used in conjunction with the CANCEL, SUBMIT, and OUTPUT commands.

Requesting an attention interrupt after issuing a STATUS command might terminate that command's processing. In this case, you cannot resume STATUS processing by pressing the Enter key as you can after most attention interrupts.

STATUS command syntax



STATUS command operand

(*jobname* (*jobid*))

Specifies the name of the batch job for which you want to know the status. If two or more jobs have the same job name, the system displays the status of all the jobs encountered and supplies job IDs for identification. When more than one job name is included in the list, the list must be enclosed within parentheses. When you specify a list of job names, you must separate the job names with standard delimiters. By default, if you do not specify any job names, you receive the status of all batch jobs in the system whose job names consist of your user ID and one identifying character (alphabetic, numeric, or one of the special characters #, $, or @). The processing might be different if your installation has replaced the default IBM-supplied installation exit.

The optional job ID subfield can consist of 1 to 8 alphanumeric characters. The first character must be alphabetic or one of the special characters (#, $, or @). The job ID is a unique job identifier assigned by the job entry subsystem (JES) at the time the job was submitted to the batch system. Currently, the only valid forms of job identifiers (*jobid*) assigned by JES are:

- JOBnnnnn or Jnnnnnnn - Jobs
- STCnnnnn or Snnnnnnn - Started Tasks
- TSUnnnnn or Tnnnnnnn - TSO Users

Note: The STATUS command uses the user ID as the jobname when building a JOB card, if not supplied.

STATUS command return codes

Table 51 on page 265 lists the return codes of STATUS command.

Table 51: STATUS command return codes

Return code	Meaning
0	Processing successful.

Table 51: STATUS command return codes (continued)	
Return code	**Meaning**
12	Processing unsuccessful. An error message has been issued.

SUBMIT command

Use the SUBMIT command to submit one or more batch jobs for background processing. Background processing is explained in *z/OS TSO/E User's Guide*.

SUBMIT is a foreground-initiated-background (FIB) command. You must be authorized by installation management to use SUBMIT. This command is generally used with the CANCEL, STATUS, and OUTPUT commands.

Users can submit jobs containing only TSO/E commands to execute commands in the background. If a job contains the LOGON command and SUBMIT finds it before encountering a JOB statement, SUBMIT uses the LOGON command to build the JOB and EXEC statements. If your installation uses security labels and security checking, you can specify LOGON in the job stream and include the SECLABEL operand. LOGON builds a JOB statement that contains the security label from the LOGON command. Using the SECLABEL operand lets users submit a job with a different security label than the security label they used to log on to TSO/E.

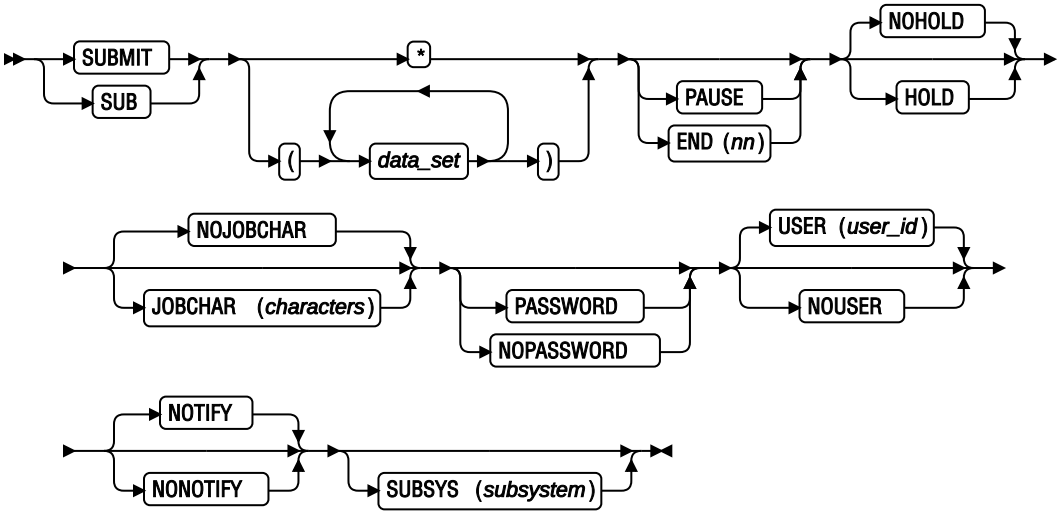
Users also can submit jobs to run at a different security label than the one they logged on with by specifying the SECLABEL operand on the JOB statement. The SECLABEL that is specified must be one that the user has access to.

If a job does not contain the LOGON command or JOB statement with a security label, the job runs at the security label the user logged on with.

If users meet certain RACF requirements, they can define another user to submit their jobs for them. When this method of submitting jobs, called surrogate job submission, is used, jobs are submitted from the second (surrogate) user's TSO/E ID. The jobs run as if submitted by the first user. For more information about surrogate job submission, see *z/OS Security Server RACF General User's Guide*.

Requesting an attention interrupt after issuing a SUBMIT command might terminate that command's processing. In this case, you cannot resume SUBMIT processing by pressing the Enter key as you can after most attention interrupts.

SUBMIT command syntax



SUBMIT command operands

(*data_set*) | *

(*data_set*)
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, separate them with delimiters, and enclose them in parentheses.

***
An asterisk (*) specifies that the job stream is to be obtained from the current source of input (for example, the terminal or currently executing CLIST). TSO/E commands can be entered directly without creating and editing a data set.

Note: All characters in the job stream are converted to uppercase before being processed.

This positional operand and the *data_set* positional operand are mutually exclusive. Either of both operands is required.

The SUBMIT * function described here is not available in EDIT mode. Job stream input received directly from the terminal or any other source will not be saved after the job is submitted. The SUBMIT * subcommand of EDIT continues to select the current data set as the input job stream. See the SUBMIT subcommand of EDIT for more information.

If the submitted job contains a job statement, the SUBMIT operands that generate job statements are ignored. The SUBMIT operands do not override the job statement.

Note: When TSO/E processes a job in a CLIST that uses the SUBMIT * command, statements following the DD * statement are left adjusted to column 1 which removes leading spaces. (This is unique to CLIST processing only; it is not a batch concern.) See *z/OS TSO/E CLISTs*, for a procedure that preserves the leading spaces in a CLIST.

PAUSE | END(*nn*)

PAUSE
Specifies that you want to make a decision after the job stream has been read in. This decision is to either continue the SUBMIT * process or terminate. If this operand is omitted, the job stream is processed when the end of the job stream is detected. The default is not to pause when the end of the job stream is reached. If you have not specified PAUSE and you subsequently make an error, the only way the submission can be aborted is with an attention interrupt. This is an optional operand.

Pause is valid only when * (asterisk) is specified for the positional parameter and you are not in EDIT mode.

END(*nn*)
Specifies a 1- or 2-character string to indicate the end of the job stream. Only alphabetic, numeric, or the special characters #, $, or @ are valid END characters. If this operand is not specified, a null or blank line indicates the end of the job stream. Specifying this operand allows blank lines to be part of the job stream. To terminate the job stream, the END character(s) must begin in column 1 and be the only data on the input line. The END character string is not considered part of the job stream. END is valid only when * (asterisk) is specified for the positional parameter and when you are not in EDIT mode.

HOLD | NOHOLD

HOLD
Specifies SUBMIT is to have job output held for use with the OUTPUT command by defaulting to the held MSGCLASS supplied by the installation manager for the user. Output directed to DD statements is held if SYSOUT=* or HOLD=YES is specified on the DD statement.

NOHOLD
Specifies job output is not to be held. NOHOLD is the default.

JOBCHAR(*characters*) | NOJOBCHAR

JOBCHAR(*characters*)
Specifies characters to be appended to the jobname on every JOB statement in the data set being submitted. Use 1 character if you plan to use the STATUS command and your job name is your user ID.

NOJOBCHAR
Specifies SUBMIT is to prompt you for jobname characters whenever the job name is the user ID. If prompting is not possible, the jobname character defaults to the letter X. NOJOBCHAR is the default. The user ID is determined by certain rules. See the USER operand for a list of the rules.

PASSWORD | NOPASSWORD

PASSWORD
Specifies a PASSWORD operand is to be inserted on the generated JOB statement by SUBMIT, if RACF is installed. SUBMIT prompts you to enter the password value in print-inhibit mode, if the terminal supports the feature. This operand is not required if a generated JOB statement or RACF is not installed. If RACF is installed, PASSWORD is the default. The password used is:

- The password (if executing in the foreground) entered on the LOGON command initiating the foreground session. The current password is used for RACF-defined users. If you have updated your password using the LOGON command, you must enter the PASSWORD operand with the new password on the SUBMIT command.
- The password on the LOGON command (if executing in the background) specified in the submitted data set. If a LOGON command is not in the data set, the USER and PASSWORD operands are not to be included on the generated JOB statement.

NOPASSWORD
Specifies the PASSWORD and USER operands are not included on the generated JOB statement. If RACF is not installed, NOPASSWORD is the default.

USER(*user_id*) | NOUSER

USER(*user_id*)
Specifies a USER operand is to be inserted on the generated JOB statement, if RACF is installed. The user ID specified is also used as the jobname for the generated JOB statement. For job name or user ID comparison for NOJOBCHAR processing, see the NOJOBCHAR operand description.

If USER or NOUSER is not entered and RACF is installed, USER is the default. The default user ID used is determined by the following rules. The rules are ordered. If the first rule is met, then that user ID is used.

1. The user ID specified on a LOGON command in the data set being submitted.
2. The user ID specified on the LOGON command (if executing in the foreground) initiating the foreground session; the user ID specified on the USER operand (if executing in the background, RACF-defined users only) on the JOB statement initiating the background session.
3. The default user ID SUBMITJB is used.

Note: If a password is not specified, the USER operand is not generated on the job statement. You can specify a password:

- On the user's SUBMIT command
- On the LOGON command data set being submitted
- In the LOGON for the current session, when executing in the foreground, by requesting that the password be stored in the TSB by using the LOGON exit

NOUSER
Specifies generated JOB statements do not include USER and PASSWORD operands. If USER is not specified and RACF is not installed, NOUSER is the default.

NOTIFY | NONOTIFY

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 is placed in the broadcast data set. You must then enter LISTBC to receive the message. If a JOB statement is generated, NOTIFY is the default.

When you supply your own JOB statement, use the NOTIFY=user_id keyword on the JOB statement if you want to be notified when the job terminates. SUBMIT ignores the NOTIFY keyword unless it is generating a JOB statement.

If NOTIFY or NONOTIFY is not specified, the default is:

- The NOTIFY operand (if executing in the foreground) is inserted on the generated JOB statement.
- The NOTIFY operand (if executing in the background) is only inserted on the generated JOB statement for RACF-defined users who have specified the USER operand on the JOB statement initiating the background session.

NONOTIFY
Specifies a termination message is not to 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.

SUBSYS(*subsystem*)
Specifies the subsystem where the submitted job runs.

SUBMIT command return codes

Table 52 on page 269 lists the return codes of SUBMIT command.

Table 52: SUBMIT command return codes

Return code	Meaning
0	Processing successful.
12	Processing unsuccessful. An error message has been issued.

SUBMIT command examples

Example 1

Operation: Submit two jobs for batch processing.

Known:

- The data sets that contain the jobs: ABTJQ.STRESS.CNTL and ABTJQ.STRAIN.CNTL

```
submit (stress, strain)
```

Example 2

Operation: Concatenate and submit data sets as a single job.

Known:

- The data set that contains the JCL for the job is JCL.CNTL(ASMFCLG)
- The data set that contains the input data is MYDATA.DATA

```
submit (jcl(asmfclg) mydata)
```

This command causes a single background job to be submitted and simultaneously concatenates a generated job card (if required), JCL, and the data. Each data set is not submitted as a separate job.

Note: The SUBMIT command uses the user ID as the jobname when building a JOB card, for an 8 character ID, if not supplied.

TERMINAL command

Use the TERMINAL command to define operating characteristics that depend primarily upon the type of terminal that you are using. You can specify the ways that you want to request an attention interruption and you can identify hardware features and capabilities. The TERMINAL command allows you to request an attention interruption whether your terminal has a key for attention interrupt.

Note: The TERMINAL command is primarily for line mode type devices that are *not* in full-screen applications. Typically TERMINAL has no effect on full-screen devices such as 3270, nor does TERMINAL work if the user is in a full-screen application, such as ISPF.

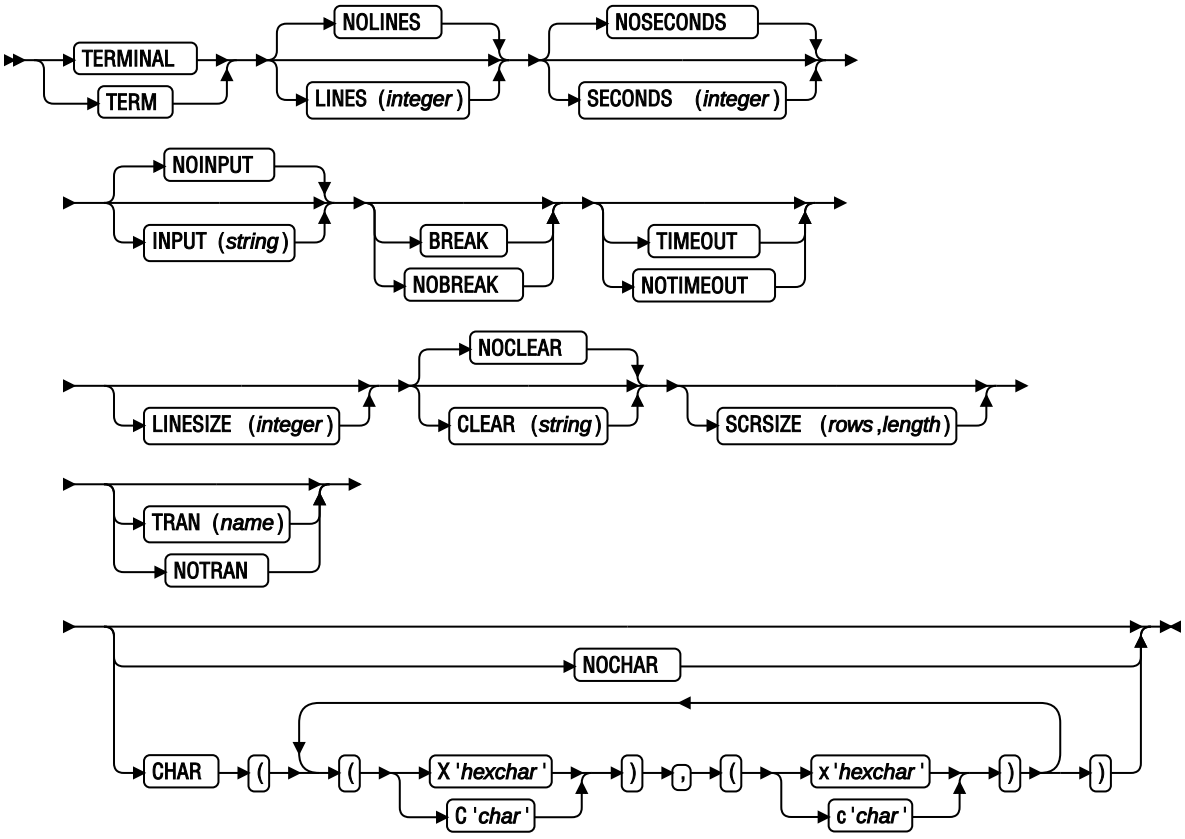
Note: The TERMINAL command is not allowed as a TSO/E command in the background.

The terminal characteristics that you have defined 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 remain in effect during the subsequent session.

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

Note: If an invoked program issues the VTAM STTRAN macro that affects the same hex value that the TERMINAL command changed, then the value set by the TERMINAL command is no longer in effect.

TERMINAL command syntax



TERMINAL command operands

LINES(*integer*) | NOLINES

LINES(*integer*) [6]
specifies an integer from 1 to 255 that indicates you want the opportunity to request an attention interruption after the specified number of lines of continuous output has been directed to your terminal.

NOLINES [6]
specifies output line count is not to be used for controlling an attention interruption. This is the default condition.

SECONDS(*integer*) | NOSECONDS

SECONDS(*integer*) [6]
specifies an integer from 10 to 2550 (in multiples of 10) to indicate that you want the opportunity to request an attention interruption after a 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 is changed to the next largest multiple of 10.

NOSECONDS [6]
specifies elapsed time is not to be used for controlling an attention interruption. This is the default condition.

INPUT(*string*) | NOINPUT

INPUT(*string*) [6]
specifies the character string, if entered as input, will cause an attention interruption. The string must be the only input entered and cannot exceed 4 characters in length.

NOINPUT [6]
specifies no character string will cause an attention interruption. This is the default condition.

BREAK | NOBREAK

BREAK
specifies, for IBM 3767 and IBM 3770 terminals, the system can interrupt your input. For other terminals, it specifies that your terminal keyboard is to 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. Because use of BREAK with a terminal type can result in loss of output or error, check with your installation system manager before specifying this operand.

Note: If a command processor for a display device is operating in full-screen mode, VTAM treats the device as if it were operating in NOBREAK mode. For a more detailed description, see *z/OS TSO/E Programming Services*.

NOBREAK
specifies, for IBM 3767 and IBM 3770 terminals, the system is not allowed to interrupt you (break in) with a message when you are entering data. For other terminals, it specifies that your terminal keyboard is to be unlocked only when your program or a command you have used requests input.

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

TIMEOUT | NOTIMEOUT

TIMEOUT [6]
specifies your terminal keyboard will lock automatically after approximately 9 to 18 seconds of no input.

NOTIMEOUT [6]
specifies your terminal keyboard will not lock automatically after approximately 9 to 18 seconds of no input.

[6] Not supported with terminals that use VTAM.

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. The integer must not exceed 255. LINESIZE is not applicable to the IBM 3270 display stations. The default values are:

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

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

If you use LINESIZE to adjust the line length of an LU1 device, the line length defaults to zero.

CLEAR(*string*) | NOCLEAR

CLEAR(*string*) [6]
specifies a character string, if entered as input, causes the screen of an IBM 3270 Display Station to be erased. The string must be the only input entered and cannot exceed 4 characters in length.

NOCLEAR [6]
specifies that you do not want to use a sequence of characters to erase the screen of an IBM 3270 Display Station. This is the default condition.

SCRSIZE(*rows*,*length*)
specifies the screen dimensions of an IBM 3270 Display Station, an LU2 device with VTAM, and a Network Terminal Option (NTO) terminal. When you specify the SCRSIZE operand, you must use the LINESIZE operand to get continuous writing on a NTO terminal.

If you are running under Session Manager, the system ignores SCRSIZE.

rows
specifies the maximum number of lines of data that can appear on the screen.

length
specifies the maximum number of characters in a line of data displayed on the screen.

Standard screen sizes (in rows and length) are:

- 6,40
- 12,40
- 12,80
- 15,64
- 24,80
- 27,132
- 32,80
- 43,80

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

TRAN(*name*) | NOTRAN

TRAN(*name*)
specifies a load module that contains tables used to translate specific characters you type at the terminal into different characters when they are seen by TSO/E. Conversely, when these characters are sent by TSO/E to the terminal, they are retranslated. Translation of numbers and uppercase letters is not allowed.

Character translation is especially useful when you are using a correspondence keyboard and would like to type the characters: <, >, |.

They are not available on a correspondence keyboard. For example, translation tables make it possible for you to specify that when you type the characters: [,], !.

TSO/E interprets them as <, >, and |.

NOTRAN
specifies no character translation is to take place.

CHAR | NOCHAR

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

NOCHAR
specifies all character translations previously specified by CHAR are no longer in effect.

TERMINAL command return codes

Table 53 on page 273 lists the return codes of TERMINAL command.

Table 53: TERMINAL command return codes

Return code	Meaning
0	Processing successful.
12	Processing unsuccessful. An error message has been issued.

TERMINAL command examples

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

Known:

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

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

TEST command

Use the TEST command to test a program, command processor, or APPC/MVS transaction program for proper execution and to locate programming errors. For APPC/MVS transaction programs, use this command to test standard transaction programs. However, you can also use this command to partially test a multi-trans type transaction program up to the point where it issues GETTRANS for the next transaction. To use the TEST command and subcommands, you should be familiar with the Assembler language and addressing conventions. See the appropriate publications for information about how to code assembler programs and definitions of assembler language terminology. Also, see *z/OS TSO/E Programming Guide*, for more information about using the TEST command and the TEST tutorial. For best results, the program to be tested should be written in Assembler language. To use the symbolic names feature of TEST, your program should have been assembled and link-edited with the TEST operands.

If the tested program attempts to LOAD, LINK, XCTL, or ATTACH another module, the module is being searched for in the following sequence: Libraries specified by an active TSOLIB command, then TASKLIB, STEPLIB, JOBLIB, LPA, and then LNKLST. If the module is not in any of these areas, it is not found. To avoid this, bring the module into virtual storage by using the LOAD subcommand of TEST.

If you enter the TEST command with operands, a pseudo or automatic breakpoint is established at +0 for the problem program being invoked under TEST. Therefore, do not use the AT subcommand of TEST (AT +0).

If you use the TEST command to test inbound APPC/MVS transaction programs, the following restrictions apply:

- You should log on with a LOGON procedure that does not allocate APPC/MVS conversations. If APPC/MVS conversations exist, TEST issues message `IKJ57501I TEST END DUE TO ERROR +` with a second-level message that explains the error and then TEST terminates. You must then perform cleanup for the conversations. (To clean up the conversations, log off. When you log on again, you should ensure that any LOGON procedure command that you specified on the logon panel does not invoke a CLIST or REXX exec that allocates APPC/MVS conversations. Note that allocation of DFM data sets might cause APPC/MVS conversations to be allocated.)
- The user-level transaction program profile for the LUs that are to be used for transaction program testing must be allowed. To allow user-level transaction program profiles, the LUADD statement in PARMLIB member APPCPMxx must include the TPLEVEL(USER) keyword. Use the LU or BASELU keyword to specify the LU on which to test the transaction program. These keywords are valid only when you use the TP keyword operand on inbound APPC/MVS transaction programs. BASELU is the default. For information about transaction program profiles, see *z/OS MVS Planning: APPC/MVS Management*.
- If your installation uses RACF and security label checking has been activated, transaction programs cannot be tested under LU=LOCAL environment. For more information about the environment for testing transaction programs, see *z/OS MVS Programming: Writing Transaction Programs for APPC/MVS*.

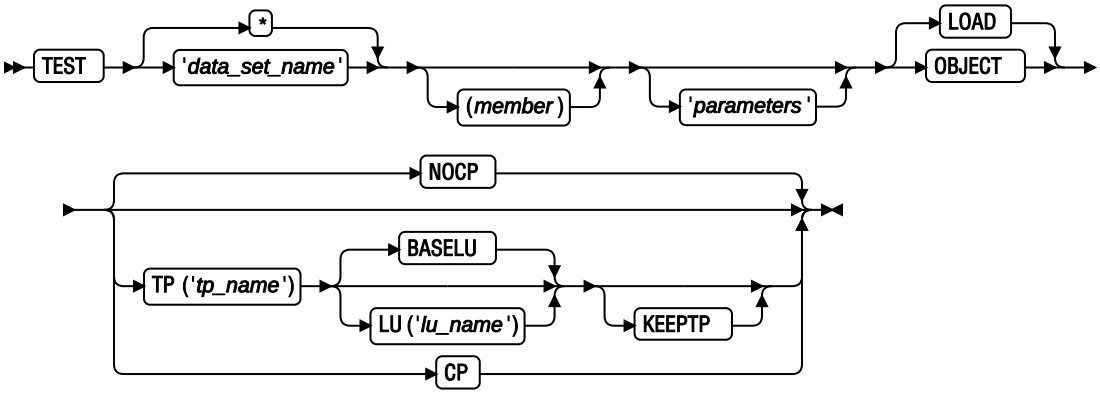
See *z/OS TSO/E Programming Guide* for information about:

- Using the TEST command. It contains a step-by-step tutorial on how to use TEST.
- Testing an APPC/MVS transaction program.
- Addressing conventions associated with TEST.
- Restrictions on the use of symbols.
- Programming considerations and restrictions for using TEST. These include:
  - 31-bit addressing
  - Using virtual fetch services
  - Cross-memory environment
  - The vector facility

Note: Requesting an attention interrupt while testing a password protected data set might terminate the TEST command's processing.

Restriction: TEST allows a user to test AMODE 24 or AMODE 31 programs. Testing programs with any other AMODE has unpredictable results. TSO TEST does not allow testing of RMODE 64 programs, but 64-bit addresses can be encountered while running under TEST. If the address 7FFFFBAD appears in a message while operating in the TEST environment, it is usually denoting the address is a 64-bit address and resides above the bar.

TEST command syntax



TEST command operands

'*data_set_name*'
specifies the name of the data set containing the program to be tested. The program must be a load module that is a member of a partitioned data set (PDS), a member of a partitioned data set extended (PDSE), or it must be an object module. A data set name must be specified to test a program that is not currently active. A currently active program is one that has abnormally terminated or has been terminated by an attention interruption.

When specifying the data set name for TEST, the name should be enclosed by single quotation marks or the LOAD or OBJECT qualifier is added to the name specified.

*****
specifies that the program to be tested resides in the standard search libraries (TSOLIB (TASKLIB), STEPLIB or JOBLIB, or current LNKLST concatenation).

Modules residing in the LPA can NOT be TESTed using the * operand.

Modules residing both in the LPA and a library in the standard search order can NOT be TESTed using the * operand.

(member)
if no member name for a partitioned data set or * is given, member TEMPNAME is assumed.

If TEST is specified with a data set name, or *, registers 2 through 12 are initialized to X'FFFFFFFF'. this allows you to determine which registers have been changed by the tested program.

When TEST is specified for a load module in a PDS or a program object in a PSDE, the program being tested can invoke other user programs, if they are members of the same PDS or PDSE. The services by which one member can invoke another in the same PDS or PDSE include LINK, LOAD, XCTL, and ATTACH.

CAUTION: The program to be tested should not have the name TEST or the name of any existing TSO/E service routine.

'*parameters*'
specifies a list of parameters to be passed to the program being tested. The 'parameters' are valid only with the NOCP or TP keywords. If you specify the CP operand, the system ignores the parameters. The list must not exceed 100 characters, including delimiters.

LOAD | OBJECT

LOAD
specifies the named program is either:

- a load module that has been processed by the linkage editor or binder and is a member of a partitioned data set (PDS)
- a program object that has been processed by the DFSMS/MVS* binder service and is a member of a partitioned data set extended (PDSE).

If both LOAD and OBJECT are omitted, LOAD is the default.

OBJECT
specifies the named program is an object module that has not been processed by the linkage editor or the DFSMS/MVS binder service. The program can be contained in a sequential data set or a member of a partitioned data set.

If OBJECT is specified on the TEST command, the tested program will be named TEMPNAME.

CP
specifies the named program is a command processor.

NOCP
specifies that the named program is not a command processor. If you do not specify CP, TP, or NOCP, then NOCP is the default.

TP('*tp_name*')
specifies that the named program is an inbound APPC/MVS transaction program. *tp_name* specifies the name of the transaction program you want to test. It is case sensitive and required if you specify the TP keyword. The *tp_name* can have a length of 1 to 64 characters consisting of uppercase and lowercase letters A–Z, numerals 1–9, and 19 special characters: `.<(+&*);-/,%_>?:'="`. If *tp_name* contains an apostrophe, you must enter two successive apostrophes for each single apostrophe.

The TEST command does not recognize transaction program alias names. For example, if you specify an alias of a transaction program name, it is considered a new transaction program name.

LU('*lu_name*') | BASELU

LU('*lu_name*')
specifies which LU is to be used. The LU keyword is valid only when the TP keyword is specified. *lu_name* specifies the LU name. The name is required if you specify LU. The name must be in uppercase and enclosed in single quotation marks.

BASELU
specifies the base LU for the user address space to be used. The BASELU keyword is valid only when the TP keyword is specified. The default is BASELU if both LU and BASELU are omitted. For more information about the base LU, see *z/OS MVS Programming: Writing Transaction Programs for APPC/MVS*.

KEEPTP
specifies that TEST should not clean up the transaction program and its conversations when TEST terminates. If you do not specify this keyword, the transaction program and its conversations are cleaned up when TEST terminates. If you specify this keyword, TEST will not clean up the transaction program and its remaining conversations.

TEST command return codes

Table 54 on page 277 lists the return codes of TEST command.

Table 54: TEST Command return codes	
Return code	**Meaning**
0	TEST is active.
4	TEST is inactive.

TEST command examples

Example 1

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

Known:

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

```
test
```

Example 2

Operation: Invoke a program for testing.

Known:

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

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

```
test payer.load(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 prefix in the user's profile is TLC55.
- The program is an object module and is not a command processor.

```
test payload object
```

Example 4

Operation: Test a command processor.

Known:

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

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

or

```
test cmds.load(output) cp
```

Note: You will be prompted to enter a command for the command processor. TSO/E prompts you for the commands you want to test.

Example 5

Operation: Invoke a command processor for testing.

Known:

- The name of the data set containing the command processor is TLC55.LOAD(OUTPUT).
- The prefix in the user's profile is TLC55.

```
test (output) cp
```

Example 6

Operation: Test an APPC/MVS transaction program.

Known:

- The TLC55.APPCTP.LOAD(myprog) data set contains the load module for the transaction program to be tested.
- MAIL is the transaction program name that the inbound allocate request will try to allocate.

```
test appctp.load(myprog) tp('MAIL') keeptp
```

Note: Because the LU keyword is not specified, TEST uses the base LU for testing. Also, the transaction program and its remaining conversations are not cleaned up by TEST when TEST terminates because the KEEPTP word is specified. See *z/OS TSO/E Programming Guide*, for more information about testing an APPC/MVS transaction program.

Example 7

Operation: Test an APPC/MVS transaction program with a specific LU.

Known:

- The TLC55.APPCTP.LOAD(myprog) data set contains the load module for the transaction program to be tested.
- MAIL is the transaction program name that the inbound allocate request will try to allocate.
- LUA is specified as the LU on which the transaction program is to be tested.

```
test appctp.load(myprog) tp('MAIL') lu('LUA')
```

Note: LUA is the LU used for testing. Also, the transaction program and its remaining conversations are cleaned up by TEST when TEST terminates because the KEEPTP keyword is not specified. See *z/OS TSO/E Programming Guide*, for more information about testing an APPC/MVS transaction program.

TEST subcommands (overview)

The following are TSO/E commands you can use in the TEST environment:

```
ALLOCATE   EXEC       LISTALC    LISTDS     RENAME   SUBMIT
ATTRIB     HELP       LISTBC     PROFILE    SEND     TERMINAL
CANCEL     LINK       LISTCAT    PROTECT    STATUS   UNALLOC (FREE)
```

The preceding commands are described with the TEST subcommands in alphabetical order. For a complete description of the syntax and function of those TSO/E commands that you can use in the TEST environment, see the corresponding TSO/E command.

Use the various TEST subcommands to perform the following basic functions:

- Execute the program from its starting address or from any address within the program.

- Display selected areas of the program as they currently appear in virtual storage, or display the contents of any of the registers. With the exception that access registers cannot be specified for indirect addressing or address expressions, you can use access registers as you need to general registers.
- Interrupt the program at specified locations. After you have interrupted the program, you can display areas of the program or any of the registers, or you can issue other subcommands of TEST to be executed before returning control to the program being tested.
- Change the contents of specified program locations in virtual storage or the contents of specific registers.

For a discussion on how to use these basic functions, see *z/OS TSO/E Programming Guide*. The subcommands of the TEST command and the TSO/E commands you can use in the TEST environment are listed in Table 56:

Table 55: Subcommands and functions of the TEST command

Subcommand	Function
ALLOCATE	Dynamically allocates the data sets required by a program intended for execution.
AND	Performs a logical AND operation on data in two locations, placing the results in the second location specified.
ASSIGNMENT OF VALUES(=)	Modifies values in virtual storage and in registers.
AT	Establishes breakpoints at specified locations.
ATTRIB	Builds a list of attributes for non-VSAM data sets, which are to be dynamically allocated.
CALL	Initializes registers and initiates processing of the program at a specified address using the standard subroutine linkage.
CANCEL	Halts processing of batch jobs submitted for the terminal.
COPY	Moves data.
DELETE	Deletes a load module from virtual storage.
DROP	Removes symbols established by the EQUATE command from the symbol table of the module being tested.
END	Terminates all operations of the TEST command and the program being tested.
EQUATE	Adds a symbol to the symbol table and assigns attributes and a location to that symbol.
EXEC	Executes a CLIST or REXX exec.
FREEMAIN	Frees a specified number of bytes of virtual storage.
GETMAIN	Acquires a specified number of bytes of virtual storage for use by the program being processed.
GO	Restarts the program at the point of interruption or at a specified address.
HELP	Lists the subcommands of TEST and explains their function, syntax, and operands.
LINK	Invokes the binder or the linkage editor service program.
LIST	Displays the contents of a virtual storage area or registers.

Table 55: Subcommands and functions of the TEST command (continued)

Subcommand	Function
LISTALC	Displays a list of the names of data sets allocated during the current TSO/E session.
LISTBC	Displays a listing of the contents of the broadcast data set or a user log data set, which contains messages of general interest (NOTICES) and messages directed to a particular user (MAIL).
LISTCAT	Lists catalog entries by name or entry type; lists selected fields for each entry.
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.
LISTDS	Displays attributes of specific data sets at the terminal.
LISTMAP	Displays a map of the user's virtual storage.
LISTPSW	Displays a program status word (PSW).
LISTTCB	Lists the contents of the current task control block (TCB). You can specify the address of another TCB.
LISTVP	Displays the partial sum number and the vector section size of a vector machine.
LISTVSR	Displays the vector status register (VSR).
LOAD	Loads a program into virtual storage for execution.
OFF	Removes breakpoints.
OR	Performs a logical OR operation on data in two locations, placing the results in the second location specified.
PROFILE	Establishes, changes, or lists the user profile.
PROTECT	Controls unauthorized access to a non-VSAM data set.
QUALIFY	Establishes the starting or base location for resolving symbolic or relative addresses; resolves identical external symbols within a program.
RENAME	Changes the name of a non-VSAM cataloged data set or a member of a partitioned data set (PDS) or creates an alias for a member of a PDS.
RUN	Terminates TEST and completes execution of the program.
SEND	Sends a message to another terminal user or to the system operator.
SETVSR	Sets fields in the vector status register.
STATUS	Displays status of batch jobs at terminal.
SUBMIT	Submits one or more batch jobs for processing.
TERMINAL	Defines the operating characteristics for the terminal being used.
UNALLOC	Frees data sets under TSO/E TEST. Because FREE is an alias for the FREEMAIN subcommand, use UNALLOC to free files under TEST.
WHERE	Displays the virtual address of a symbol or entry point, or the address of the next executable instruction. WHERE can also be used to display the module and CSECT name and the displacement into the CSECT corresponding to an address.

TEST—ALLOCATE command

Use the ALLOCATE command to dynamically allocate the data sets required by a program intended for execution. For a description of the ALLOCATE command syntax and function, see the "ALLOCATE command" on page 8.

TEST—AND subcommand

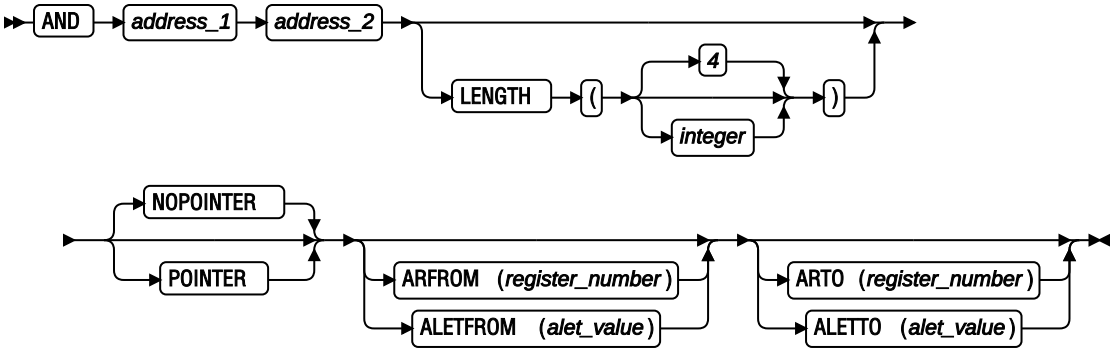
Use the AND subcommand to perform a logical AND operation on data or addresses from:

- One virtual storage address to another
- One general register to another
- A general register to virtual storage
- Virtual storage to a general register
- An access register to virtual storage
- Virtual storage to an access register
- One access register to another

The AND subcommand can be used to:

- Alter the contents of the general registers
- AND an entire data field with another

TEST—AND subcommand syntax



TEST—AND subcommand operands

address_1
specifies the location of data that is to be ANDed with data pointed to by *address_2*.

If you do not specify POINTER and there is a breakpoint in the data pointed to by *address_1*, the TSO/E TEST processor terminates the AND operation.

address_2
specifies the location of the data that is to be ANDed with data pointed to by *address_1*. When the AND operation is complete, the result is stored at this location.

You can specify *address_1* and *address_2* as:

- An absolute address
- A symbolic address
- A relative address
- An indirect address
- An address expression

- A module name and entry name (separated by a period)
- A general register
- An entry name (preceded by a period)
- An access register

ARTO(*register_number*)
specifies that the location of the data pointed to by *address_2* is in an alternate address/data space referred to by an access register. Valid access register numbers are 0 through 15. The operands ARTO and ALETTO (or ALTO) are mutually exclusive.

ARFROM(*register_number*)
specifies that the location of the data pointed to by *address_1* is in an alternate address/data space referred to by an access register. Valid access register numbers are 0 through 15. The operands ARFROM, ALETFROM, and POINTER are mutually exclusive.

ALETTO(*alet_value*) | ALTO(*alet_value*)
specifies that the location of the data pointed to by *address_2* is in an alternate address/data space. The ALETTO value may be from 1 to 8 hexadecimal characters. The operands ALETTO and ARTO are mutually exclusive.

ALETFROM(*alet_value*) | ALFROM(*alet_value*)
specifies that the location of the data pointed to by *address_1* is in an alternate address/data space. The ALETFROM value may be from 1 to 8 hexadecimal characters. The operands ALETFROM, ARFROM and POINTER are mutually exclusive.

LENGTH(*integer*) | LENGTH(4)
specifies the length, in decimal, of the field to be copied. If an integer is not specified, LENGTH defaults to 4 bytes. The maximum length is 256 bytes.

POINTER
specifies *address_1* is to be validity checked to see that it does not exceed maximum virtual storage size. *address_1* is then treated as an immediate operand (hexadecimal literal) with a maximum length of 4 bytes (that is, an address converted to its hexadecimal equivalent). When using the POINTER operand, do not specify a general register as *address_1*. The POINTER operand and the operands ARFROM and ALETFROM are mutually exclusive.

NOPOINTER
specifies *address_1* is to be treated as an address. If neither POINTER nor NOPOINTER is specified, NOPOINTER is the default.

The AND subcommand treats the 16 general registers as contiguous fields. The user can AND 10 bytes from general register 0 to another location as follows:

```
and 0R 80060. length(10)
```

The AND subcommand ANDs the 4 bytes of register 0, the 4 bytes of register 1, and the high-order 2 bytes of register 2 to virtual storage beginning at location 80060. When a register is specified as *address_1*, the maximum length of data that is ANDed is the total length of the general registers or 64 bytes.

TEST—AND subcommand examples

Example 1

Operation: AND two full words of data each in a virtual storage location placing the result in the second location.

Known:

- The starting address of the data to be used as the first operand: 80680
- The starting address of the data to be used as the second operand and the location of the result: 80690

```
and 80680. 80690. length(8)
```

Example 2

Operation: AND the contents of two registers, placing the result in the second register specified.

Known:

- The register which contains the data specified as the first operand: 10
- The register which contains data specified as the second operand and the result: 5

```
and 10r 5r
```

Example 3

Operation: Turn off the high-order bit of a register.

Known:

- The AND value: X'7F'
- The register: 1

```
and 7F. 1r l(1) pointer
```

Note: Specifying the pointer operand causes 7F to be treated as an immediate operand and not as an address.

Example 4

Operation: AND 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 ANDed: 14
- The virtual storage location which is to contain the second operand and result: 80680
- The length of the data to be ANDed: 8 bytes

```
and 14r% 80680. l(8) nopoint
```

Example 5

Operation: AND a fullword with X'7F' into the storage where general register 3 points in the alternate address/data space referred to by the ALET 00010004.

```
and 7f. 3r? pointer aletto(00010004)
```

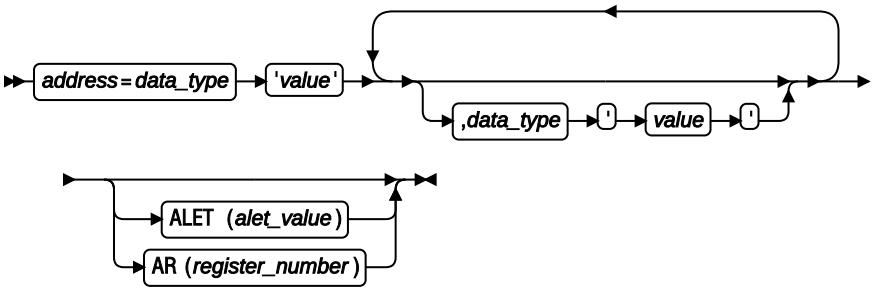
Assignment of values function of TEST

Use the assignment function to change:

- The contents of specified program locations in virtual storage
- The contents of specific registers
- The contents of storage in an alternate address/data space

When processing is halted at a breakpoint or before execution is initiated, you can modify values in virtual storage and in registers. This function is implicit; that is, you do not enter a subcommand name. The system performs the function in response to operands that you enter.

Syntax of values function of TEST



Operands of values function of TEST

address
specifies the location that you want to contain a new value. You can specify *address* as:

- An absolute address
- A symbolic address
- A relative address
- An indirect address
- An address expression
- A module name and entry name (separated by a period)
- An entry name (preceded by a period)
- A general register
- A floating point register
- A vector register
- A vector register element
- An access register
- An alternate address/data space
- The vector mask register

***data_type* '*value*'[, *data_type* '*value*',...]**
specifies the type of data and the value that you want to place in the specified location. If you want to specify more than one *data_type*, enclose the list in parentheses, for example, `(data_type 'value',data_type 'value')`. You indicate the type of data by one of the following codes:

Code	Type of data	Maximum length (bytes) *	Storage boundary
			Data types must begin on specified boundary for a virtual storage address
C	Character	One line of input, continued lines permitted	C-byte
X	Hexadecimal	64	X-byte
B	Binary	64	B-byte
H	Fixed point binary (halfword)	6	H-halfword
F	Fixed point binary (fullword)	11	F-fullword
E	Floating point (single precision)	13	E-fullword
D	Floating point (double precision)	22	D-doubleword
P	Packed decimal	32	P-byte
Z	Zoned decimal	17	Z-byte

Code	Type of data	Maximum length (bytes) *	Storage boundary
A	Address constant	11	A-fullword
S	Address (base + displacement)	8	S-halfword
Y	Address constant (halfword)	6	Y-halfword

* All characters within the quotation marks are included in the length.

Following is a list of valid entries and syntax for data type:

C
‘character value’

X
‘hexadecimal value’

B
‘binary value’

H
‘[±] decimal value’

The minimum value for H-type is -32768 and the maximum value is 32767

F
‘[±] decimal value’

The minimum value for F-type is -2147483648 and the maximum value is 2147483647

E
‘[+] decimal value [E[+] decimal exponent]’

A maximum of eight digits is allowed for the decimal value and a maximum of two digits is allowed for the decimal exponent

D
‘[+] decimal value [E[+] decimal exponent]’

A maximum of 17 digits is allowed for the decimal value and a maximum of two digits is allowed for the decimal exponent

P
‘[+] decimal value’

A maximum of 31 digits is allowed

Z
‘[+] decimal value’

A maximum of 16 digits is allowed

A
‘[±] decimal value’

The minimum decimal value is -2147483648 and the maximum decimal value is 2147483647

S
‘decimal value(register number)’

The decimal value can be from 0 to 4095 and the register number must be from 0 to 15 (decimal form)

Y
‘[+] decimal value’

The decimal value may be from 0 to 32767

You include your data following the code. Your data must be enclosed within apostrophes. Any single apostrophes within your data must be coded as two single apostrophes. Character data can be entered. If necessary, all other data types will be translated into uppercase.

A list of data can be specified by enclosing the list in parentheses. The data in the list is stored at the beginning of the location specified by the address operand.

Values assigned to general registers and access registers are placed in registers right-justified and padded with binary zeroes.

When a virtual storage address is assigned a list of *data_type* values, the address must reside on the appropriate boundary for the specified *data_type* of the first value. Storage bytes for subsequent *data_type* values will be skipped to align data on the appropriate boundary for the data type requested.

If the length of the value you assign to the vector mask register is greater than the length of the vector mask register, an error message is issued. If the length of the value is shorter than the vector mask register, the value is placed in the vector mask register left-justified, and the remaining bits are unchanged.

The following restrictions apply to general registers, floating-point registers, vector registers, access registers and the vector mask register.

1. Specify only one *data_type* for floating-point registers. Additional *data_type*s are ignored.
2. Assign only X or E *data_type*s to single precision floating-point registers.
3. Assign only X, F, or E *data_type*s to single precision vector registers.
4. Assign only X or D *data_type*s to double precision floating-point registers.
5. Assign only X or D *data_type*s to double precision vector registers.
6. With the exception of the D-type of data, general registers and access registers can be assigned any *data_type*
7. Assign only X or B *data_type*s to the vector mask register.

When a general register, floating point register, vector register, or vector register element is assigned a list of *data_type*'values', the first value is assigned to the specified register or register element. Subsequent *data_type* values are assigned to contiguous higher-numbered registers or register elements. If register 15 is reached and *data_type* values remain, the values are wrapped around to register 0 and subsequent registers, if needed. For more information about programming considerations for using the vector registers, see *z/OS TSO/E Programming Guide*.

If data is assigned to a storage area that contains a breakpoint, the breakpoint is removed and a warning message is displayed at the terminal.

ALET(*alet_value*)
specifies the alternate address/data space where you want to change storage. You can specify from 1 to 8 hexadecimal characters to represent the *alet_value*.

AR(*register_number*)
specifies the access register that contains the alet to be used to determine where you want to change storage. Valid access register numbers are 0 through 15.

Examples of values function of TEST

Example 1

Operation: Insert a character string at a particular location in virtual storage.

Known:

- The address is a symbol: INPOINT
- The data: January 1, 1985

```
inpoint=c'january 1, 1985'
```

Example 2

Operation: Insert a binary number into a register.

Known:

- The number of the register: register 6
- The data: 0000 0001 0110 0011

```
6r=b'0000000101100011'
```

Example 3

Operation: Initialize registers 0 through 3 to zeroes and register 15 to 4.

```
15R=(x'4',x'0',x'0',x'0'x'0')
```

Note: The sixteen (16) general registers are treated as contiguous fields with register 0 immediately following register 15.

Example 4

Operation: Assign a new base and displacement for an instruction that was found to be in error.

Known:

- LA instruction at +30 is X'41309020'. In this instruction, the current base register is 9 and the displacement is a decimal value of 32 (hexadecimal value of 20). The base register should be 10 and the decimal displacement should be 98 (hexadecimal value of 62).

```
+32=S'98(10)'
```

After this assignment, the instruction at +30 is X'4130A062'.

Example 5

Operation: Insert a number in packed format at a particular address in virtual storage.

Known:

- Absolute address: C3D41, decimal value to be packed is -1038.

```
c3d41.=p'-1038'
```

Example 6

Operation: Set the entire contents of the vector register 1 to hexadecimal zeros.

```
1v(*)=X'00000000'
```

Example 7

Operation: Set the tenth element in vector register 1 to decimal 33.

```
1v(10)=f'33'
```

Example 8

Operation: Set elements 3 and 4 of vector register 3 to X'00' and X'02'.

```
3v(3)=(X'00',X'02')
```

Example 9

Operation: Set the first element of vector registers 0 and 1 to the double precision floating point value of +33E+2.

```
0w(1)=d'+33E+2'
```

Example 10

Operation: Assign the value 100 to the four bytes at the address pointed to by register 9. The storage for addressing is in the address space referred to by the ALET value 9E00.

```
9r?=F'100'ALET(9E00)
```

Example 11

Operation: Set the contents of access register 7 to zeros.

```
7a=x'00000000'
```

Example 12

Operation: Set the vector mask register to the hexadecimal value 046C471F.

```
0M=x'46C471F'
```

TEST—AT subcommand

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.

If you enter the TEST command with any operands, a pseudo or automatic breakpoint is established at +0 for the problem program being invoked under TEST. Therefore, do not specify AT +0.

If you set a breakpoint following a fullscreen TPUT macro and preceding a TGET macro, the fullscreen message is overlaid by the TEST line mode message (IKJ57024I). For more information, see *z/OS TSO/E Programming Guide*.

You cannot establish a breakpoint at:

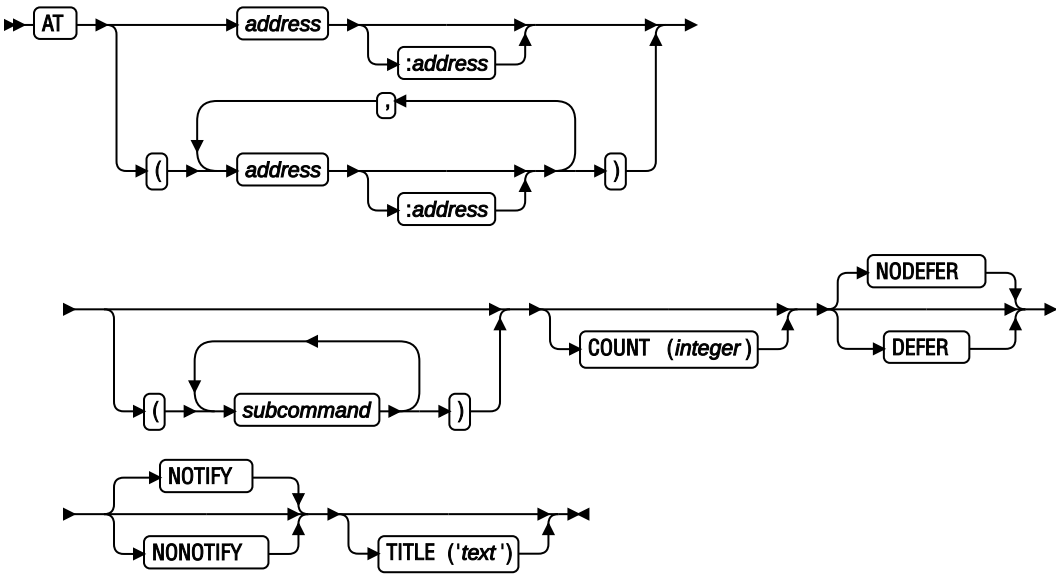
- The target of an execute instruction or the execute instruction itself.
- An instruction that is to be modified by the execution of other in-line code before the execution of the breakpoint.
- A user-written SVC exit.
- An instruction that other code depends upon to be the same. See “Example 7” on page 291.

For some instructions, the AT subcommand establishes a single pass breakpoint that is removed automatically after the breakpoint hits. These instructions include:

- all branch relative instructions (such as, BRC, BRAS, or BRCT).
- all space-switching instructions (such as, PC, SAC, SACF, SSAR, PT, or PR).
- any instructions that save the current PSW address except BAL, BALR, BAS, BASR, BSM, or BASSM.

A single pass breakpoint is removed automatically during execution of the GO subcommand. After resuming execution at the breakpoint, the breakpoint is no longer in effect until you establish it again by using the AT subcommand. If you want to establish the breakpoint, you must use the AT subcommand after you issue the GO subcommand, not before.

TEST—AT subcommand syntax



TEST—AT subcommand operands

address
specifies a location that is to contain a breakpoint. 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 must be on a halfword boundary. A breakpoint is established at each instruction between the two addresses. When a range of addresses is specified, assignment of breakpoints halts when a not valid instruction is encountered.

(*address*)
specifies several addresses that are to contain breakpoints. Each address must be on a halfword boundary. The list must be enclosed within parentheses, and the addresses in the list must be separated by standard delimiters (one or more blanks or a comma). A breakpoint is established at each address.

For *address*, *address*:*address*, (*address*), specify *address* as:

- An absolute address
- A symbolic address
- A relative address
- An indirect address
- An address expression
- A module name and entry name (separated by a period)
- An entry name (preceded by a period)

subcommands
specifies one or more subcommands to be executed when the program is interrupted at the indicated location. If you specify more than one subcommand, the subcommands must be separated by semicolons. The list cannot be longer than 255 characters. **If a CLIST is executed as part of the subcommand list the results of the execution may not occur in the expected order.**

Note: If an OFF subcommand in the list removes the breakpoint for which a list is specified, all remaining subcommands in that list are ignored.

COUNT(*integer*)
specifies that processing is not to be halted at the breakpoint until it has been encountered the specified number of times. This operand is directly applicable to program loop situations where an

instruction is executed several times. Processing is halted each time the breakpoint has been encountered the number of times specified for the integer operand. The integer specified cannot exceed 65,535.

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

DEFER
specifies the breakpoint is to be established in a program that is *not yet* in virtual storage. The program to contain the breakpoint is brought in as a result of a LINK, LOAD, ATTACH, or XCTL macro instruction by the program being tested. When you specify this operand, you must qualify the address of the breakpoint:

```
MODULENAME.ENTRYNAME.RELATIVE
```

or

```
MODULENAME.ENTRYNAME.SYMBOL
```

All breakpoint addresses listed in an AT subcommand with the DEFER operand must refer to the same load module.

NOTIFY
specifies that if the breakpoint is encountered, it will be identified at the terminal. NOTIFY is the default.

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

TITLE(‘*text*’)
specifies from 1 to 50 characters of text displayed following the word AT whenever the tested program stops at the breakpoint associated with that text. The text is intended to serve as a meaningful identification of the instruction address at which the program stops. It is used instead of an address. If NONOTIFY is specified, nothing is displayed.

A list of addresses can be associated with the same text and the text is displayed whenever the associated breakpoint is reached. If a range is specified and TITLE (‘*text*’) is listed as an operand, the text is displayed in the form: ‘*text_string*’ + displacement. Displacement is the hexadecimal offset at the breakpoint encountered from the beginning of the range.

Note: If your program is running in supervisor state or in a PSW protection key less than 8, breakpoints are ignored.

TEST—AT subcommand examples

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 the section that you want to test: 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. When the interruption occurs, you want a meaningful identification at the breakpoint.

Known:

- The address for the breakpoint: 15R%

```
at 15r% count(10) title('entry after 10 loops')
```

Example 4

Operation: Establish a breakpoint for a program that is not presently in virtual storage.

Known:

- The name of the load module: CALCULAT
- The CSECT name: INTEREST
- The symbolic address for the breakpoint: TOTAL

```
at calculat.interest.total defer
```

Example 5

Operation: Have the following subcommands executed when the breakpoint at TAC is reached: LISTTCB PRINT(TCBS), LISTPSW, and GO CALCULAT

```
at tac (listtcb print(tcbs);listpsw;go calculat)
```

Example 6

Operation: Request that the following subcommands be executed when the breakpoint at symbol NOW is reached: LISTMAP, LISTTCB, OFF NOW, AT +32, and GO.

```
at now (listmap;listtcb;off now;at +32;go)
```

The last two subcommands will not be executed because the breakpoint (NOW) and its subcommand list will have been removed.

Example 7

Operation: Do not set a breakpoint at an instruction that other code depends upon to be unchanged.

```
                            WAIT   ECB=ECBX, LONG=YES
0000  4110  C020    LA      1,ECBX          load parameter reg. 1
0004  4100  0001     LA      0,1(0,0)           count omitted, 1 used
0008  0780              BCR     8,0                gives an inline '80'
000A  BF08  C009  ICM      0,8,*-1           insert into hi-byte
000E  0A01           SVC      1                 link to wait routine
:
0020                      ECBX    DS               F
```

In this assembler coding example, the instruction at +A causes the high-order byte of register 0 to contain an ‘80’. Inserting a breakpoint at +8 causes the instruction at +8 to replace the inline ‘80’ produced by the WAIT macro with an SVC 97.

TEST—ATTRIB command

Use the ATTRIB command to build a list of attributes for non-VSAM data sets that are to be dynamically allocated. For a description of the ATTRIB command syntax and function, see the “ATTRIB command” on page 53.

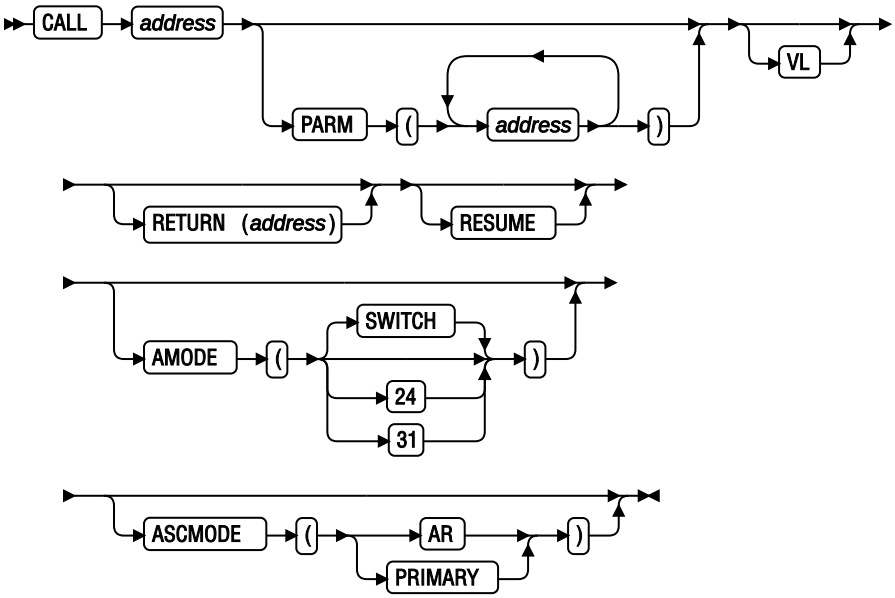
TEST—CALL subcommand

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 “Example 2” on page 296 and “Example 3” on page 296 under “TEST—COPY subcommand” on page 294).

The CALL subcommand of TEST places the return address of the tested program in register 14. The high-order bit of register 14 is set to reflect the addressing mode of the tested program.

TEST—CALL subcommand syntax



TEST—CALL subcommand operands

address
specifies the address where processing is to begin. Register 15 contains this address when the program under test begins execution. You can specify *address* as:

- An absolute address
- A symbolic address
- A relative address
- An indirect address
- An address expression
- A module name and entry name (separated by a period)
- An entry name (preceded by a period)

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

VL
specifies the high-order bit of the last fullword of the list of addresses pointed to by general register 1 is to be set to one.

RETURN(*address*)
specifies on completion of execution, the called program returns control to the address in register 14. The high-order bit of register 14 reflects the addressing mode of the tested program before the issuance of the CALL subcommand. If RETURN is omitted, register 14 contains the address of a breakpoint instruction.

RESUME
specifies upon completion of execution, the called program returns control to the address of the last breakpoint before the CALL.

AMODE [(24 | 31 | SWITCH)]
specifies the addressing mode in which the called program begins execution. If AMODE(SWITCH) is specified, the called program continues execution in the addressing mode that is non-current when CALL is issued. You can determine the current addressing mode by issuing the LISTPSW command. If AMODE is not specified, there is no change in addressing mode.

ASCMODE(AR | PRIMARY)
specifies the PSW mode in which the called program executes. If you specify ASCMODE(PRIMARY), the PSW mode is set to execute the program using the primary address space control mode (in primary mode). When ASCMODE(AR) is specified, the PSW is set to execute the program in AR mode.

TEST—CALL subcommand examples

Example 1

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

Known:

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

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

Note: The following message is issued after completion of the called routine:

```
'IKJ57023I PROGRAM UNDER TEST HAS TERMINATED NORMALLY+'
```

This message is then issued because no return address was specified. If GO is now specified without an address, the TEST session is terminated.

Example 2

Operation: Initiate® execution at a particular location.

Known:

- The starting address: STARTBD
- The addresses of data to be passed: BDFLAGS PRFTTBL COSTTBL ERREXIT
- Set the high-order bit of the last address parameter to 1 so that the program can tell the end of the list. After execution, control is to be returned to: +24A

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

Example 3

Operation: Initiate execution at label COMPUTE and have execution begin at label NEXT when control is returned by register 14.

```
call compute return(next)
```

TEST—CANCEL command

Use the CANCEL command to halt processing of batch jobs submitted from the terminal. For a description of the CANCEL command syntax and function, see the "CANCEL command" on page 65.

TEST—COPY subcommand

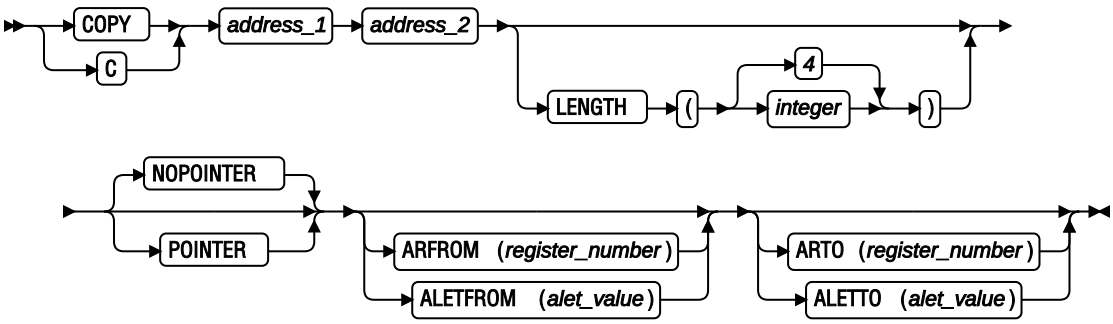
Use the COPY subcommand to transfer data or addresses from:

- One storage address to another
- One general register to another
- A general register to virtual storage
- Virtual storage to a general register
- An access register to virtual storage
- Virtual storage to an access register
- One access register to another

In addition, you can use the COPY subcommand 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

TEST—COPY subcommand syntax



TEST—COPY subcommand operands

address_1
specifies a location that contains data to be copied.

address_2
specifies a location that receives the data after it is copied.

You can specify *address_1* and *address_2* as:

- An absolute address
- A symbolic address
- A relative address

- An indirect address
- An address expression
- A module name and entry name (separated by a period)
- A general register
- An access register

ARFROM(*register_number*)
specifies that the location of the data pointed to by *address_1* is in an alternate address/data space referred to by the specified access register. Valid access register numbers are 0 through 15. The operands ARFROM, ALETFROM, and POINTER are mutually exclusive.

ALETFROM(*alet_value*) | ALFROM(*alet_value*)
specifies that the location of the data pointed to by *address_1* is in an alternate address/data space. The ALETFROM value may be from 1 to 8 hexadecimal characters. The operands ALETFROM, ARFROM, and POINTER are mutually exclusive.

ARTO(*register_number*)
specifies that the location of the data pointed to by *address_2* is in an alternate address/data space referred to by an access register. Valid access register numbers are 0 through 15. The operands ARTO and ALETTO (or ALTO) are mutually exclusive.

ALETTO(*alet_value*) | ALTO(*alet_value*)
specifies that the location of the data pointed to by *address_2* is in an alternate address/data space. The ALETTO value may be from 1 to 8 hexadecimal characters. The operands ALETTO and ARTO are mutually exclusive.

LENGTH(*integer*) | LENGTH(4)
specifies the length, in decimal, of the field to be copied. If an integer is not specified, LENGTH defaults to 4 bytes. The maximum length is 65,535 bytes in a storage-to-storage copy operation and 64 bytes when a register is specified.

POINTER
specifies *address_1* is to be validity checked to see that it does not exceed maximum virtual storage size. *address_1* is then 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 transferred into the location specified by *address_2*. When using the POINTER operand, do not specify a general register as *address_1*. POINTER and the operands ARFROM and ALETFROM are mutually exclusive.

NOPOINTER
specifies *address_1* is to be treated as an address, not as an immediate operand. NOPOINTER is the default.

The COPY subcommand treats the 16 general registers as contiguous fields. You can specify that 10 bytes be moved from general register 0 to another location.

```
copy 0r 80060. length(10)
```

The COPY subcommand moves the 4 bytes of register 0, the 4 bytes of register 1, and the high-order 2 bytes of register 2 to virtual storage beginning at location 80060. When a register is specified as *address_1*, the maximum length of data transferred is the total length of the general registers or 64 bytes.

When the value of *address_2* is one greater than *address_1*, propagation of the data in *address_1* occurs. When the value of *address_2* is more than one greater than the value of *address_1*, no propagation occurs.

TEST—COPY subcommand examples

Example 1

Operation: Transfer two full words of data from one virtual storage location to another.

Known:

- The starting address of the data: 80680

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

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

Example 2

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

Known:

- The register which contains the data to be copied: 10
- The register which contains the data to be received: 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
```

Note: Specifying the pointer operand causes 80 to be treated as an immediate operand and not as an address.

Example 6

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

Known:

- The module name and the entry_point name: IEFBR14.IEFBR14
- The desired virtual storage location: STARTPTR

```
c iefbr14.iefbr14 startptr p
```

Example 7

Operation: Copy the contents of an area pointed to by a register into another area.

Known:

- The register which points to the area that contains the data to be moved: 14
- The real storage location which is to contain the data: 80680
- The length of the data to be moved: 8 bytes

```
c 14r% 80680. l(8) nopoint
```

Example 8

Operation: Copy the 72 bytes where register 13 points in the primary address space to location 1000 in the address space referred to by access register 5.

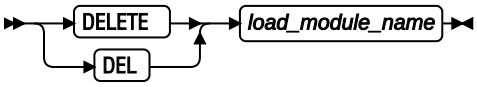
```
copy 13r? 1000.  arto(5) length(72)
```

TEST—DELETE subcommand

Use the DELETE subcommand to delete, from virtual storage, a load module that was loaded by the tested program, or by one of its subtasks.

Use the DELETE subcommand to delete a module that was loaded above or below 16MB by the tested program or by the LOAD subcommand of TEST.

TEST—DELETE subcommand syntax



TEST—DELETE subcommand operand

load_module_name
specifies the name of the load module to be deleted. The load name is the name (which might be an alias) by which the program is known to the system when it is in virtual storage. The name must not exceed 8 characters.

TEST—DELETE subcommand examples

Example 1

Operation: Delete a load module from virtual storage.

Known:

- The name of the load module: TOTAL

```
delete total
```

or

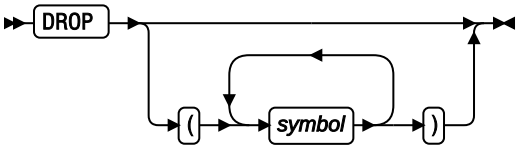
```
del total
```

TEST—DROP subcommand

Use the DROP subcommand to remove symbols from the symbol table of the module being tested. You can only remove symbols that you established with the EQUATE subcommand or the EQUATE operand of

the GETMAIN subcommand. You cannot remove symbols that were established by the linkage editor. If the program being tested was assembled with the TEST option and the EQUATE subcommand was used to override the location and type of the symbol within the program, then when the DROP subcommand is used to delete that symbol from the symbol table, the symbol will reflect the original location and type within the program.

TEST—DROP subcommand syntax



TEST—DROP subcommand operand

(symbol)
specifies one or more symbols that you want to remove from the symbol table created by the EQUATE subcommand or the EQUATE operand of the GETMAIN subcommand. When you specify only one symbol, you do not have to enclose the symbol within parentheses. However, two or more symbols must be enclosed by parentheses. If you do not specify any symbols, the entire table of symbols is removed.

TEST—DROP subcommand examples

Example 1

Operation: Remove all symbols that you have established with the EQUATE subcommand.

```
drop
```

Example 2

Operation: Remove a symbol from the symbol table.

Known:

- The name of the symbol: DATE

```
drop date
```

Example 3

Operation: Remove several symbols from the symbol table.

Known:

- The names of the symbols: STARTADD TOTAL WRITESUM

```
drop (startadd total writesum)
```

TEST—END subcommand

Use the END subcommand to terminate all functions of the TEST command and the program being tested.

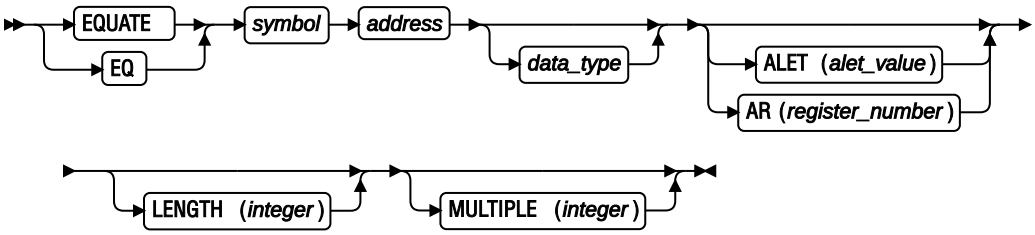
TEST—END subcommand syntax

The END subcommand does not close an opened data set. Use the GO subcommand to close an opened data set. Normal exit cleanup procedures should also be used.

TEST—EQUATE subcommand

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 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. A symbol within DSECT can be accessed if the DSECT name is defined using the EQUATE subcommand. You can also modify the data attributes (type, length, and multiplicity); use the EQUATE subcommand to modify attributes of existing equated symbols. The DROP subcommand removes symbols added by the EQUATE subcommand. Symbols established by the EQUATE subcommand are defined for the duration of the TEST session only.

TEST—EQUATE subcommand syntax



TEST—EQUATE subcommand operands

symbol
specifies the symbol (name) that you want to add to the symbol table so that you can refer to an address symbolically. The symbol must consist of 1 to 8 alphanumeric characters, the first of which is an alphabetic character.

address
specifies the address is to equate to the symbol that you specified. You can specify *address* as:

- An absolute address
- A symbolic address
- A relative address
- An indirect address
- An address expression
- A module name and entry name (separated by a period)
- An entry name (preceded by a period)

data_type
specifies the characteristics you want to attribute to the data at the location given by address. These might be the same as the original characteristics. Indicate the type of data by one of the following codes:

Code	Type of data	Maximum length (bytes)
C	Character	256
X	Hexadecimal	256
B	Binary	256
I	Assembler instruction	256
H	Fixed point binary (halfword)	8
F	Fixed point binary (fullword)	8
E	Floating point (single precision)	8
D	Floating point (double precision)	8
P	Packed decimal	16

Code	Type of data	Maximum length (bytes)
Z	Zoned decimal	16
A	Address constant	4
S	Address (base + displacement)	2
Y	Address constant (halfword)	2

ALET(*alet_value*)
specifies the alternate address/data space that the EQUATEd variable can reference. You can specify from 1 to 8 hexadecimal characters to represent the *alet_value*.

AR(*register_number*)
specifies the access register that contains the alet used to determine the alternate address/data space that the EQUATEd variable can reference. Valid access register numbers are 0 through 15.

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 apply:

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

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

If you do not specify any operands, the defaults are:

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

If both multiplicity and length are specified for *data_type* I, the multiplicity is ignored.

TEST—EQUATE subcommand examples

Example 1

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

Known:

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

```
equate exitrtn total+4
```

Example 2

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

Known:

- The symbol: CONSTANT
- The new address: 1FAA0

- The new attributes: type: C, length: L(8), multiplicity: M(2)

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

Example 3

Operation: Add the symbol NAMES to the symbol table to access a list of 6 names. Each name is 8 characters long.

Known:

- The names are stored one after the other at relative address +12C.

```
equate names +12c 1(8) m(6) c
```

Example 4

Operation: Add SYMBOL1 to the symbol table. SYMBOL1 represents the location 3000 in the address/data space referred to via ALET 00010003.

```
equate symbol1  3000.  alet(00010003)
```

TEST—EXEC command

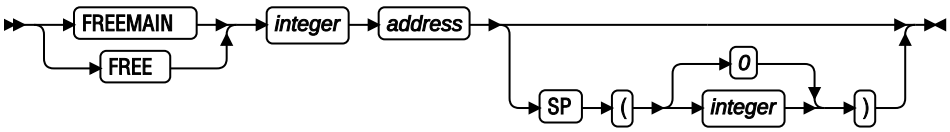
Use the EXEC command to execute a CLIST or REXX exec. For a description of the EXEC command syntax and function, see the “EXEC command” on page 120.

Specify only REXX statements in the REXX exec. Specify only TEST subcommands and CLIST statements in the CLIST. You cannot specify TSO/E commands in the CLIST or REXX exec until you specify END or RUN to terminate TEST.

TEST—FREEMAIN subcommand

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

TEST—FREEMAIN subcommand syntax



TEST—FREEMAIN subcommand operands

integer
specifies the number of decimal bytes of virtual storage to be released.

address
specifies the location of the space to be freed. It must be a multiple of 8 bytes.

Use the LISTMAP subcommand to help locate previously acquired virtual storage.

You can specify *address* as:

- An absolute address
- A symbolic address
- A relative address
- An indirect address
- An address expression

- A module name and entry name (separated by a period)
- An entry name (preceded by a period)

SP(*integer*) | SP(0)
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 0 through 127.

TEST—FREEMAIN subcommand examples

Example 1

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

Known:

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

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

Example 2

Operation: Free space in virtual storage that was previously obtained by a GETMAIN subcommand.

Known:

- The size of the space: 100 decimal bytes
- The address of the space to be freed: X'A4' past the address in register 3
- The space to be freed: in subpool 0

```
freemain 100 3r%+A4
```

Example 3

Operation: Free subpool 127.

```
freemain 0 0. sp(127)
```

Attention: Do not attempt to free all of subpool 78. If you want to free a portion of subpool 78, be careful not to free the storage obtained by the TMP. This results in freeing the TMP's data areas because subpool 78 is shared. The deletion of the TMP portion of subpool 78 causes your session to terminate.

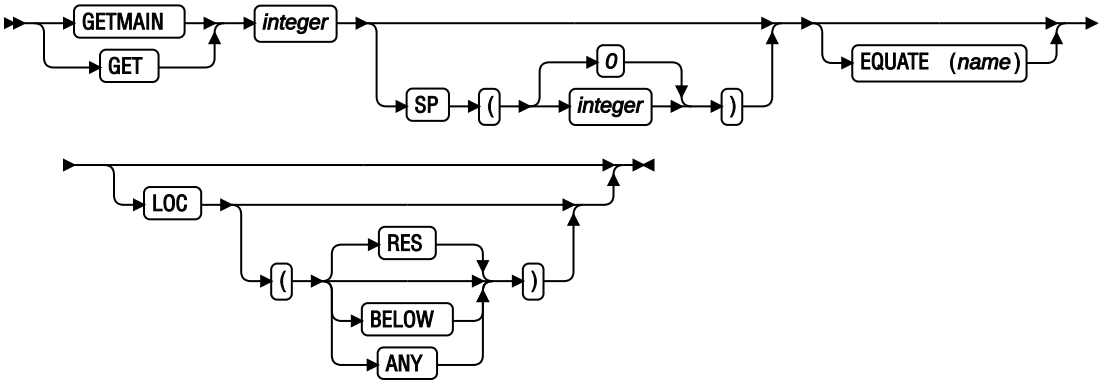
You can release an entire subpool by specifying a length of 0, an absolute address of 0, and a subpool in the range 1-127.

If you specify a non-zero address, the length must also be non-zero.

TEST—GETMAIN subcommand

Use the GETMAIN subcommand to obtain a specified number of bytes of virtual storage. The GETMAIN subcommand displays the starting address of the virtual storage obtained.

TEST—GETMAIN subcommand syntax



TEST—GETMAIN subcommand sperands

integer
specifies the number of bytes, in decimal form, of virtual storage to be obtained.

SP(*integer*) | SP(0)
specifies the number of a subpool from which the virtual storage is to be obtained. If you omit this operand, the default value is subpool zero. The integer must be in the range 0 through 127.

EQUATE(*name*)
specifies the address of acquired virtual storage is to be equated to the symbol specified by name and placed in the TEST internal symbol table.

LOC(BELOW)
specifies the virtual and real storage area must be below 16 MB.

LOC(ANY)
specifies the virtual storage area can be anywhere in the virtual storage addressing range. The actual location (above or below 16 MB) of the virtual storage area depends on the subpool specified. If the requested subpool is supported above 16 MB, GETMAIN allocates virtual storage above 16 MB, if possible.

LOC(RES)
specifies the address of the virtual storage area depends upon the residence of the next instruction to be executed. If the instruction address in the PSW for the tested program is below 16 MB, the request is processed as LOC(BELOW). If the instruction address is above 16 MB, the request is processed as LOC(ANY). LOC(RES) is the default.

TEST—GETMAIN subcommand examples

Example 1

Operation: Obtain 240 decimal bytes of virtual storage from subpool 0.

```
getmain 240
```

Example 2

Operation: Obtain 500 bytes of virtual storage from subpool 3 and equate starting address to symbolic name AREA.

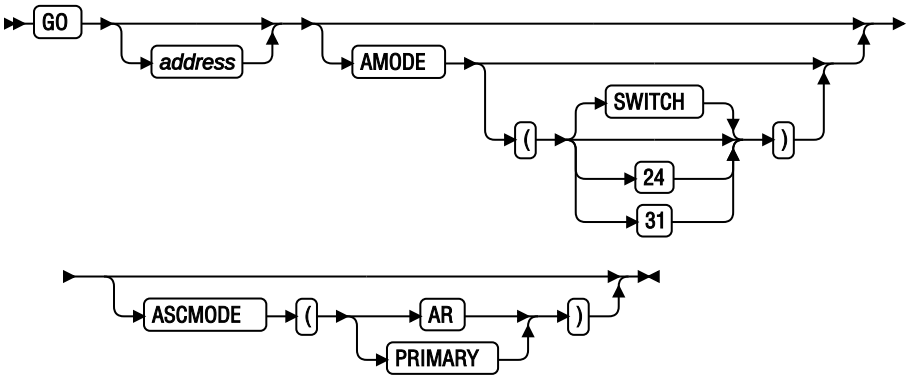
```
get 500 sp(3) equate(area)
```

TEST—GO subcommand

Use the GO subcommand to start or restart program execution from a particular address. If the program was interrupted for a breakpoint and you want to continue from the breakpoint, there is no need to specify the address. However, you can start execution at any point by specifying the address.

TSO/E TEST supports single pass breakpoints for some instructions (for example, BRC, BRAS, and BRCT.) (Refer to the AT subcommand of TEST for more information about single pass breakpoints.) A single pass breakpoint will be removed automatically during execution of the GO subcommand of TSO/E TEST. After resuming execution at the breakpoint, the breakpoint will no longer be in effect until the user reestablishes it by using the AT subcommand of TSO/E TEST. If the user desires to reestablish the breakpoint, the breakpoint must be reestablished after the 'GO' subcommand has been issued.

TEST—GO subcommand syntax



TEST—GO subcommand operands

address
specifies the address where processing is to begin. You can specify *address* as:

- A symbolic address
- A relative address
- An absolute address
- An address expression
- A module name and entry name (separated by a period)
- An entry name (preceded by a period)

When the problem program completes processing, the following message is displayed at the terminal:

```
IKJ57023I PROGRAM UNDER TEST HAS TERMINATED NORMALLY+
```

If you now issue the GO subcommand without specifying an address, the TEST session is terminated.

AMODE [(24 | 31 | SWITCH)]
specifies the addressing mode in which program execution resumes after the GO subcommand has been issued. You can specify AMODE without specifying an address. However, if the word AMODE or any abbreviation of the word AMODE is defined as a symbolic address, GO AMODE executes as follows: program execution starts at the last breakpoint and the SWITCH default is taken.

If you do not specify AMODE, there is no change in addressing mode.

ASCMODE(AR | PRIMARY)
specifies the PSW mode in which the program executes after the GO command is issued. If you specify ASCMODE(PRIMARY), the PSW is set to execute the program using the primary address space

control mode (in primary mode). Specifying ASCMODE(AR) sets the PSW to execute the program in AR mode.

TEST—GO subcommand examples

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
```

TEST—HELP command

Use the HELP command to obtain the syntax and function of the TEST subcommands. For a description of the HELP command syntax and function, see the "HELP command" on page 143.

TEST—LINK command

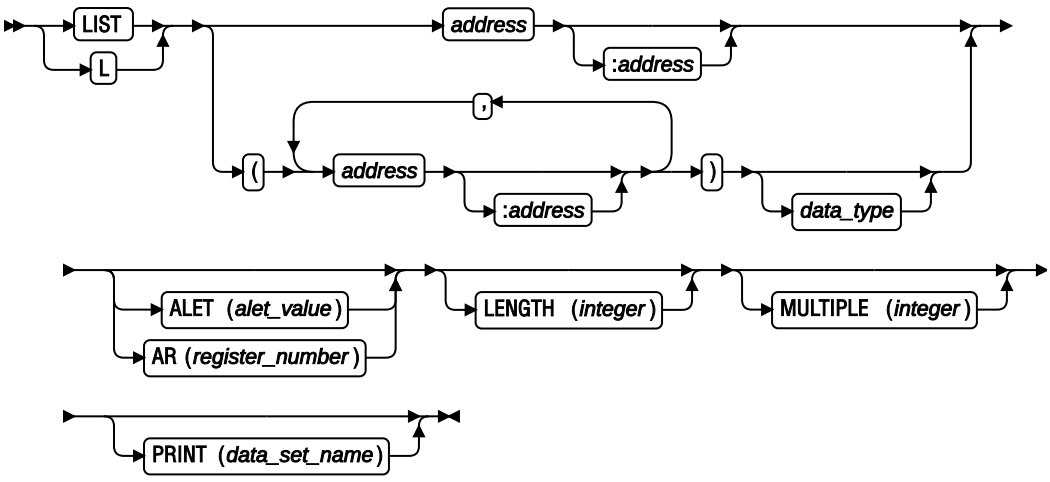
Use the LINK command to invoke the binder or the linkage editor service program. For the description of the LINK command syntax and function, see the "LINK command" on page 148.

TEST—LIST subcommand

Use the LIST subcommand to display at your terminal or place in a data set the following:

- The contents of a specified area of virtual storage
- The contents of registers or vector registers
- The contents of access registers
- Data in alternate address/data spaces that is referred to via an access register
- The vector mask register.

TEST—LIST subcommand syntax



TEST—LIST subcommand operands

address
specifies the location of data that you want displayed at your terminal or placed into a data set.

address*:*address
specifies that you want the data located between the specified addresses displayed at your terminal or placed into a data set.

(*address*)
specifies several addresses of data that you want displayed at your terminal or placed into a data set. The data at each location is retrieved. If the first address of a range is a register, the second address must also be the same type of register (floating-point, general, or vector). 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).

If a range of addresses is specified on LIST and the ending address is in fetch protected storage, you are prompted (if in PROMPT mode) to reenter the address. If you want a range of addresses, you must reenter the range, not just the ending address.

You can create a load module that contains more than one DSECT or CSECT within the same symbolic name. When you list an unqualified symbolic address in a load module, the LIST command displays the area associated with the first occurrence of the symbol. Use the fully-qualified name, 'module_name.csect.symbol_name', to display occurrences other than the first.

For *address*, *address*:*address*, (*address*), specify *address* as:

- An absolute address
- A symbolic address
- A relative address
- An indirect address
- An address expression
- A module name and entry name (separated by a period)
- An entry name (preceded by a period)
- A general register
- A floating-point register
- A vector register
- A vector register element
- An access register
- The vector mask register.

data_type
specifies the type of data that is in the specified location. Indicate the type of data using one of the following codes:

Code	Type of data	Maximum length (Bytes)
C	Character	256
X	Hexadecimal	256
B	Binary	256
I	Assembler instruction	256
H	Fixed point binary (halfword)	8
F	Fixed point binary (fullword)	8
E	Floating point (single precision)	8
D	Floating point (double precision)	8
P	Packed decimal	16

Code	Type of data	Maximum length (Bytes)
Z	Zoned decimal	16
A	Address constant	4
S	Address (base + displacement)	2
Y	Address constant (halfword)	2
XC	Hexadecimal and EBCDIC	256

All accepted *data_type*s allow the specified address to be aligned on a byte boundary even though certain *data_type*s cannot be assigned to a byte boundary. The default for *data_type* is hexadecimal.

The XC *data_type* indicates that you want to display, side-by-side, the hexadecimal and EBCDIC contents of storage. The contents are displayed in hexadecimal first, followed by EBCDIC.

A general register is displayed in decimal format if the F *data_type* is used. Otherwise, regardless of the type specified, a general register is displayed in hexadecimal. Floating-point registers are listed in floating-point format if *data_type* is not specified. However, floating-point registers can be listed in hexadecimal format by using the X *data_type*. If any *data_type* other than D, E, or X is specified for floating-point registers, *data_type* is ignored and the register is listed in floating-point format.

For vector registers, if you do not specify the *data_type*, then LIST displays them in floating-point format. You can display vector registers in hexadecimal for both single (V) and double (W) precision registers. You can also display single precision (V) registers in fixed-point binary. If you specify another data type, LIST ignores it. For more information about programming considerations for using the Vector facility, see *z/OS TSO/E Programming Guide*.

Specify 0m to display the vector mask register. It can be displayed in hexadecimal or binary format.

Access registers (A) are displayed in decimal if you specify the F *data_type*. Otherwise, they are displayed in the default *data_type*, hexadecimal.

If an area is to be displayed using the I *data_type* and the area contains a not valid op code, only the area up to that not valid op code is displayed.

ALET(*alet_value*)
specifies that the contents of storage in an alternate address/data space are to be displayed. You can specify from 1 to 8 hexadecimal characters to represent the *alet_value*.

The *alet_value* used to reference storage appears at the far right of the display of storage. If you display storage in the primary address space, the *alet_value* is zeros. If you display storage in an alternate address/data space, the *alet_value* is the hexadecimal value you specified. ALET and AR are mutually exclusive.

Note: The *alet_value* is displayed whenever storage is listed, not only when you specify the ALET or AR keywords.

AR(*register_number*)
specifies the access register number used to reference data in an alternate address/data space. Valid register numbers for AR are 0 through 15.

The *alet_value* in the access register used to reference storage appears at the far right of the display of storage. If you display storage in the primary address space, the *alet_value* is zeros. If you display storage in an alternate address/data space, the *alet_value* is the hexadecimal value of the data in the access register. AR and ALET are mutually exclusive.

Note: The *alet_value* is displayed whenever storage is listed, not only when you specify the AR or ALET keywords.

LENGTH(*integer*)
indicates the length, in bytes, of the data that is to be listed. If you use a symbolic address and do not specify LENGTH, the value for the LENGTH operand is retrieved from the internal TEST symbol table or from the length associated with a symbol in a program. Otherwise, the following default values 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
XC	4

When the *data_type* is I, either LENGTH or MULTIPLE can be specified, but not both. If both are specified, the MULTIPLE operand is ignored, but no error message is printed.

MULTIPLE(*integer*)
Use with the LENGTH operand. It gives you the following options:

- The ability to format the data to be listed (see "Example 8" on page 310).
- A way of printing more than 256 bytes at a time. The value you specify for the integer determines how many lengths or multiples of *data_type* you want listed. The value supplied for the integer cannot exceed 256.

For I type data, the value supplied for MULTIPLE defines the number of instructions to be displayed. If you use a symbolic address and do not specify either LENGTH or MULTIPLE, the length retrieved from the internal TEST symbol table or from the program is used and multiplicity is ignored.

PRINT(*data_set_name*)

specifies the name of a sequential data set to which the data is directed. If you omit this operand, the data is directed to your terminal.

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

If PRINT(*data_set_name*) is specified, use the following table to determine the format of the output.

If the *data_set_name* is not specified within quotation marks, the descriptive qualifier TESTLIST is added.

If your record type was:	Fixed, fixed blocked, or undefined		Variable or variable-blocked	
Then it is changed to variable-blocked with the following attributes:	Recordsize 125	Blocksize 1629	Recordsize 125	Blocksize 129

Record and block sizes greater than those specified in the preceding table are unchanged.

The specified data set is kept open until:

- The TEST session is ended by a RUN or END subcommand, or
- A LIST, LISTDCB, LISTDEB, LISTMAP, LISTPSW, LISTTCB, or LISTVSR subcommand is entered specifying a different PRINT data set. In this case, the previous data set is closed and the current one is opened.

Notice that "P" is not a valid truncation for the PRINT operand on the LIST subcommand because the single letter "P" may be used as a *data_type* specifying that the data to be listed is in packed decimal format. See also "Example 6" on page 309.

TEST—LIST subcommand examples

Example 1

Operation: List the contents of floating-point register 2 in single precision.

```
list  2e
```

Example 2

Operation: List all of the general registers.

```
list  0r:15r
```

Example 3

Operation: List all of the floating-point registers in double precision.

```
list  0d:6d
```

Example 4

Operation: List 20 instructions starting with address +3A

```
list +3a i m(20)
```

Example 5

Operation: List the contents of an area of virtual storage.

Known:

- The area to be displayed is between labels COUNTERA and DTABLE.
- The data is to be listed in character format for a length of 130 bytes.
- The name of the data set where the data is to be put: MYDATA.DCDUMP.

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

Example 6

Operation: List the contents of two words of storage containing packed decimal numbers, and place the output into a print data set.

Known:

- The area to be displayed starting at X'22FF4' contains two words in packed decimal format. Each packed decimal number is of a length of four bytes.
- The name of the data set where the listed data is to be placed is MYDATA.DCDUMP.
- Assume the hexadecimal contents at address X'22FF4' is X'0000135C', and the hexadecimal contents at X'22FF8' is X'0032767D'.

```
list 22FF4. p m(2) len(4) print('mydata.dcdump')
```

The following two lines where written to the print data set MYDATA.DCDUMP:

```
00022FF4.  +135
00022FF8.  -32767
```

Example 7

Operation: List the contents of virtual storage at several addresses.

Known:

- The addresses: TOTAL1, TOTAL2, TOTAL3, and ALLTOTAL
- Each address is to be displayed in fixed-point binary format in three lines of 3 bytes each.

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

Example 8

Operation: List the first six fullwords in the communications vector table (CVT).

Known:

- The absolute address of the CVT: 10
- The user is operating in TEST mode.
- The data is to be listed in hexadecimal form in six lines of 4 bytes each.

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

  ```
  qualify 10.%
  ```

- TEST: The system response

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

The display at your terminal might resemble the following:

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

Example 9

Operation: Display the entire contents of vector register 1 in hexadecimal.

```
list 1v(*) x
```

Example 10

Operation: Display the fourth element of vector register 1 in fullword fixed point binary.

```
list 1v(4) f
```

Example 11

Operation: Display elements 3 through 20 of vector register 3 in single precision floating point.

```
list 3v(3):3v(20)
```

Example 12

Operation: Display the entire contents of all 16 vector registers in single precision floating point.

```
list 0v(*):15v(*)
```

Example 13

Operation: Display the entire contents of vector register 0 in double precision floating point.

```
list 0w(*)
```

Example 14

Operation: Display elements 5 to 25 of vector register 2 in double precision floating point.

```
list 2w(5):2w(25)
```

Example 15

Operation: List the contents of storage at address 4AD8 in the address/data space referred to by access register 4.

```
list 4ad8.  ar(4)
```

Example 16

Operation: List the contents of storage at the location pointed to by general register 2 in the address/data space referred to by access register 8.

```
list 2r?  ar(8)
```

Example 17

Operation: List in decimal the contents of storage at the location pointed to by the contents of the storage pointed to by register 5. The storage for all addressing is in the address/data space referred to by access register 6.

```
list 5r??  ar(6) f
```

Example 18

Operation: List the contents of storage at location 100 in the address/data space referred to by the ALET value 00010003.

```
list 100.  alet(00010003)
```

TEST—LISTALC command

Use the LISTALC command to obtain a list of names of the data sets allocated during the current user session. For a description of the LISTALC command syntax and function, see the “LISTALC command” on page 162.

TEST—LISTBC command

Use the LISTBC command to obtain a listing of the contents of the broadcast data set or the user log data set. It contains messages of general interest (NOTICES) and messages directed to particular users (MAIL).

For a description of the LISTBC command syntax and function, see the “LISTBC command” on page 165.

TEST—LISTCAT command

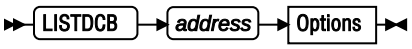
Use the LISTCAT command to list catalog entries by name of entry type and selected fields for each entry. For a description of the LISTCAT command syntax and function, see the “LISTCAT command” on page 167.

TEST—LISTDCB subcommand

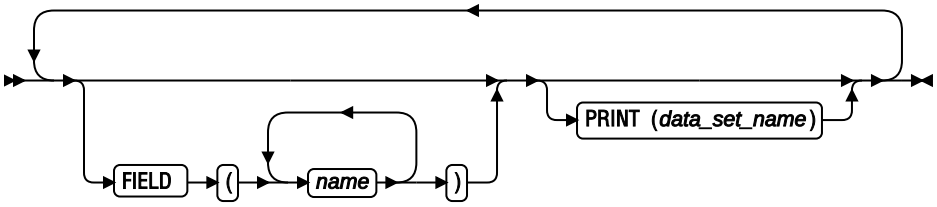
Use the LISTDCB subcommand to list the contents of a data control block (DCB). You must provide the address of the beginning of the DCB.

You can display the selected fields. 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.

TEST—LISTDCB subcommand syntax



Options



TEST—LISTDCB subcommand operands

address
specifies the address of the DCB that you want displayed. The address must be on a fullword boundary. You can specify *address* as:

- An absolute address
- A symbolic address
- A relative address
- An indirect address
- An address expression
- A module name and entry name (separated by a period)
- An entry name (preceded by a period).

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

PRINT(*data_set_name*)
specifies the name of a sequential data set to which the data is directed. If you omit this operand, the data is directed 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 they are being opened for the first time. Otherwise, they are treated as MOD data sets.

If PRINT(*data_set_name*) is specified, use the following table to determine the format of the output.

If the *data_set_name* is not specified within quotation marks, the descriptive qualifier TESTLIST is added.

If your record type was:	**Fixed, fixed blocked, or undefined**		**Variable or variable-blocked**	
Then it is changed to variable-blocked with the following attributes:	Recordsize 125	Blocksize 1629	Recordsize 125	Blocksize 129

Record and block sizes greater than those specified in the preceding table are unchanged.

The specified data set is kept open until:

- The TEST session is ended by a RUN or END subcommand, or
- A LIST, LISTDCB, LISTDEB, LISTMAP, LISTPSW, LISTTCB, or LISTVSR subcommand is entered specifying a different PRINT data set. In this case, the previous data set is closed and the current one is opened.

TEST—LISTDCB subcommand examples

Example 1

Operation: List the RECFM field of a DCB for the program that is being tested.

Known:

- The DCB begins at location: DCBIN

```
listdcb dcbin field(dcbrecfm)
```

Example 2

Operation: List an entire DCB.

Known:

- The absolute address of the DCB: A33B4

```
listdcb a33b4.
```

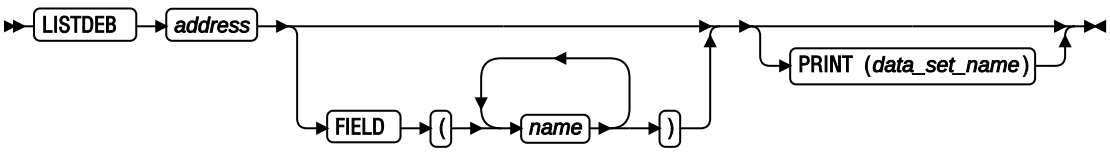
TEST—LISTDEB subcommand

Use the LISTDEB subcommand to list the contents of a data extent block (DEB). You must provide the address of the DEB.

If a copy of the control block is in extended virtual storage, the LISTDEB subcommand accepts addresses greater than 16 MB, even though the block itself will always be in virtual storage below 16 MB. Even if an absolute address has been specified, LISTDEB displays the virtual address before formatting the control block.

In addition to the 32 byte basic section of the DEB, you can receive up to 16 direct access device dependent sections of 16 bytes each, until the full length has been displayed. If you want, you can have only selected fields displayed.

TEST—LISTDEB subcommand syntax



TEST—LISTDEB subcommand operands

address
specifies the address is the beginning of the DEB. It must be on a fullword boundary. You can specify *address* as:

- An absolute address
- A symbolic address
- A relative address
- An indirect address

- An address expression
- A module name and entry name (separated by a period)
- An entry name (preceded by a period).

FIELD(*name*)
specifies one or more names of the particular fields in the DEB that you want to display at your terminal. If you omit this operand, the entire DEB is listed.

PRINT(*data_set_name*)
specifies the name of a sequential data set to which the data is directed. If you omit this operand, the data is directed 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 they are being opened for the first time. Otherwise, they are treated as MOD data sets.

If PRINT(*data_set_name*) is specified, use the following table to determine the format of the output.

If the *data_set_name* is not specified within quotation marks, the descriptive qualifier TESTLIST is added.

If your record type was:	**Fixed, fixed blocked, or undefined**		**Variable or variable-blocked**	
Then it is changed to variable-blocked with the following attributes:	Recordsize 125	Blocksize 1629	Recordsize 125	Blocksize 129

Record and block sizes greater than those specified in the preceding table are unchanged.

The specified data set is kept open until:

- The TEST session is ended by a RUN or END subcommand, or
- A LIST, LISTDCB, LISTDEB, LISTMAP, LISTPSW, LISTTCB, or LISTVSR subcommand is entered specifying a different PRINT data set. In this case, the previous data set is closed and the current one is opened.

TEST—LISTDEB subcommand examples

Example 1

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

Known:

- The address of the DEB is 44 decimal (2C hexadecimal) bytes past the beginning of the DCB.
- The address of the DEB: DCBIN+2C%

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

Example 2

Operation: List the following fields in the DEB: DEBDCBAD and DEBOFLGS

Known:

- The address of the DEB is 44 decimal (2C hexadecimal) bytes past the beginning of the DCB. The address of the DCB is in register 8.

```
listdeb 8r%+2c% field(debdcbad,deboflgs)
```

TEST—LISTDS command

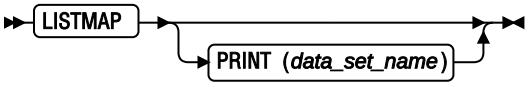
Use the LISTDS command to display attributes of specific data sets at the terminal. For a description of the LISTDS command syntax and function, see the “LISTDS command” on page 171.

TEST—LISTMAP subcommand

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

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

TEST—LISTMAP subcommand syntax



TEST—LISTMAP subcommand sperands

PRINT(*data_set_name*)
specifies the name of a sequential data set to which the data is directed. If you omit this operand, the data is directed 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 they are being opened for the first time. Otherwise, they are treated as MOD data sets.

If PRINT(*data_set_name*) is specified, use the following table to determine the format of the output.

If the *data_set_name* is not specified within quotation marks, the descriptive qualifier TESTLIST is added.

If your record type was:	Fixed, fixed blocked, or undefined		Variable or variable-blocked	
Then it is changed to variable-blocked with the following attributes:	Recordsize 125	Blocksize 1629	Recordsize 125	Blocksize 129

Record and block sizes greater than those specified in the preceding table are unchanged.

The specified data set is kept open until:

- The TEST session is ended by a RUN or END subcommand, or
- A LIST, LISTDCB, LISTDEB, LISTMAP, LISTPSW, LISTTCB, or LISTVSR subcommand is entered specifying a different PRINT data set. In this case, the previous data set is closed and the current one is opened.

TEST—LISTMAP subcommand examples

Example 1

Operation: Display a map of virtual storage at your terminal.

```
listmap
```

Example 2

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

Known:

- The name of the data set: ACDQP.MAP.TESTLIST

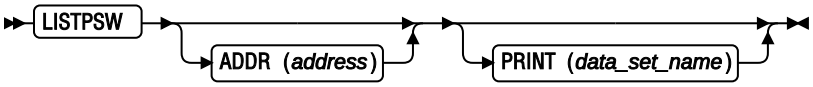
- The prefix in the user's profile: ACDQP

```
listmap print(map)
```

TEST—LISTPSW subcommand

Use the LISTPSW subcommand to display a program status word (PSW) at your terminal.

TEST—LISTPSW subcommand syntax



TEST—LISTPSW subcommand operands

ADDR(*address*)
specifies the address of a particular PSW. If you do not specify an address, you receive the current PSW for the program that is executing. You can specify *address* as:

- An absolute address
- A symbolic address
- A relative address
- An indirect address
- An address expression
- A module name and entry name (separated by a period)
- An entry name (preceded by a period).

PRINT(*data_set_name*)

specifies the name of a sequential data set to which the data is directed. If you omit this operand, the data is directed 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 they are being opened for the first time. Otherwise, they are treated as MOD data sets.

If PRINT(*data_set_name*) is specified, use the following table to determine the format of the output.

If the *data_set_name* is not specified within quotation marks, the descriptive qualifier TESTLIST is added.

If your record type was:	Fixed, fixed blocked, or undefined		Variable or variable-blocked	
Then it is changed to variable-blocked with the following attributes:	Recordsize 125	Blocksize 1629	Recordsize 125	Blocksize 129

Record and block sizes greater than those specified in the preceding table are unchanged.

The specified data set is kept open until:

- The TEST session is ended by a RUN or END subcommand, or
- A LIST, LISTDCB, LISTDEB, LISTMAP, LISTPSW, LISTTCB, or LISTVSR subcommand is entered specifying a different PRINT data set. In this case, the previous data set is closed and the current one is opened.

TEST—LISTPSW subcommand examples

Example 1

Operation: Display the current PSW at your terminal.

```
listpsw
```

Example 2

Operation: Direct the input/output old PSW into a data set.

Known:

- The prefix in the user's profile: ANZAL2
- The address of the PSW (in hexadecimal): 38
- The name of the data set: ANZAL2.PSWS.TESTLIST

```
listpsw addr(38.) print(psws)
```

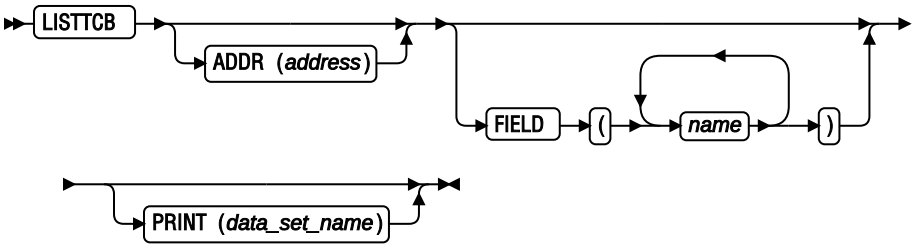
TEST—LISTTCB subcommand

Use the LISTTCB subcommand to display the contents of a task control block (TCB). You can provide the address of the beginning of the TCB.

If a copy of the control block is in extended virtual storage, the LISTTCB subcommand accepts addresses greater than 16MB, even though the block itself is below 16MB in virtual storage. Even if an absolute address is specified, LISTTCB displays the virtual address of the requested TCB before formatting the control block.

If you want, you can have only selected fields displayed.

TEST—LISTTCB subcommand syntax



TEST—LISTTCB subcommand operands

ADDR(*address*)
specifies the address must be on a fullword boundary. The address identifies the particular TCB that you want to display. If you omit an address, the TCB for the current task is displayed. You can specify *address* as:

- An absolute address
- A symbolic address
- A relative address
- An indirect address
- An address expression
- A module name and entry name (separated by a period)
- An entry name (preceded by a period).

FIELD(*name*)
specifies one or more names of the particular fields in the TCB that you want to display. If you omit this operand, the entire TCB is displayed.

PRINT(*data_set_name*)
specifies the name of a sequential data set to which the data is directed. If you omit this operand, the data is directed 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 they are being opened for the first time. Otherwise, they are treated as MOD data sets.

If PRINT(*data_set_name*) is specified, use the following table to determine the format of the output.

If the *data_set_name* is not specified within quotation marks, the descriptive qualifier TESTLIST is added.

If your record type was:	Fixed, fixed blocked, or undefined		Variable or variable-blocked	
Then it is changed to variable-blocked with the following attributes:	Recordsize 125	Blocksize 1629	Recordsize 125	Blocksize 129

Record and block sizes greater than those specified in the preceding table are unchanged.

The specified data set is kept open until:

- The TEST session is ended by a RUN or END subcommand, or
- A LIST, LISTDCB, LISTDEB, LISTMAP, LISTPSW, LISTTCB, or LISTVSR subcommand is entered specifying a different PRINT data set. In this case, the previous data set is closed and the current one is opened.

TEST—LISTTCB subcommand examples

Example 1

Operation: Direct a copy of the TCB for the current task into a data set.

Known:

- The prefix in the user's profile is NAN75.
- The name of the data set: NAN75.TCBS.TESTLIST

```
listtcb print(tcbs)
```

Example 2

Operation: Save a copy of some fields of a task's control block that is not active in a data set for future information.

Known:

- The symbolic address of the TCB: MYTCB2
- The fields that are being requested: TCBTIO TCBCMP TCBGRS
- The name of the data set: SCOTT.TCBDATA

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

Example 3

Operation: List the entire TCB for the current task.

```
listtcb
```

TEST—LISTVP subcommand

Use the LISTVP subcommand to display the partial sum number and the vector section size of a vector machine.

TEST—LISTVP subcommand syntax

TEST—LISTVP subcommand examples

Example 1

Operation: Determine the vector section size and partial sum number of the vector machine currently being used.

```
listvp
```

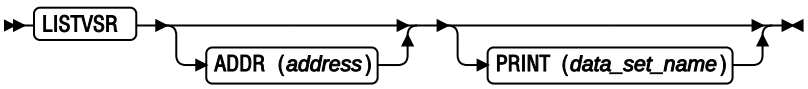
The output might look similar to the following:

```
IKJ57026I VECTOR SYSTEM PARAMETERS
SECTION SIZE:  00256[7]
PARTIAL SUM:   00004[7]
```

TEST—LISTVSR subcommand

Use the LISTVSR subcommand to display the contents of the vector status register (VSR).

TEST—LISTVSR subcommand syntax



TEST—LISTVSR subcommand operands

ADDR(*address*)
specifies the address of a particular vector status register. If you do not specify an address, you receive the current vector status register for the program that is executing. You can specify *address* as:

- An absolute address
- A symbolic address
- A relative address
- An indirect address
- An address expression
- A module name and entry name (separated by a period)
- An entry name (preceded by a period).

PRINT(*data_set_name*)
specifies the name of a sequential data set to which the data is directed. If you omit this operand, the data is directed to your terminal.

[7] This value will differ based on the machine currently used.

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

If PRINT(*data_set_name*) is specified, use the following table to determine the format of the output.

If the *data_set_name* is not specified within quotation marks, the descriptive qualifier TESTLIST is added.

If your record type was:	**Fixed, fixed blocked, or undefined**		**Variable or variable-blocked**	
Then it is changed to variable-blocked with the following attributes:	Recordsize 125	Blocksize 1629	Recordsize 125	Blocksize 129

Record and block sizes greater than those specified in the preceding table are unchanged.

The specified data set is kept open until:

- The TEST session is ended by a RUN or END subcommand, or
- A LIST, LISTDCB, LISTDEB, LISTMAP, LISTPSW, LISTTCB, or LISTVSR subcommand is entered specifying a different PRINT data set. In this case, the previous data set is closed and the current one is opened.

TEST—LISTVSR subcommand examples

Example 1

Operation: Display the contents of a vector status register after issuing a RESTORE VSR instruction (VSRRS):

```
listvsr
```

The output might look similar to the following:

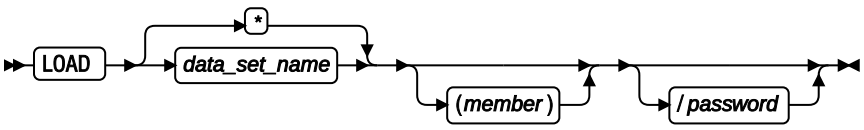
```
VSR LOCATED AT 7FFF9EF8
      RESERVED        VMM    VCT     VIX      VIU        VCH
   00000000 00000000   0    00127   00127   00000000   00000000
```

TEST—LOAD subcommand

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

Use the LOAD subcommand to load a program above or below 16MB virtual storage based on its RMODE characteristics. If the displayed entry address is greater than X'7FFFFFFF', the addressing mode is 31-bit. In this case, X'80000000' must be subtracted from the displayed number to obtain the actual address.

TEST—LOAD subcommand syntax



TEST—LOAD subcommand operands

data_set_name
specifies the name of a member of a PDS or a PDSE from which the program is to be executed. If you do not specify quotation marks around the data set name, LOAD assumes a suffix of LOAD.

*****
specifies that the program to be loaded resides in the LPA and the standard libraries are to be searched (linklist).

(*member*)
specifies the name of a member of the partitioned data set containing the module to be loaded. If the member name is not specified, TEMPNAME is used. If the *data_set_name* is not specified within quotation marks, the LOAD qualifier is added.

password
specifies the password for a password protected data set.

TEST—LOAD subcommand examples

Example 1

Operation: Load a program named GSCORES from the data set ATX03.LOAD.

Known:

- The prefix in the user's profile is ATX03.

```
load 'atx03.load(gscores)'
```

or

```
load(gscores)
```

Example 2

Operation: Load a module named ATTEMPT from data set ATX03.TEST.LOAD.

Known:

- The prefix in the user's profile is ATX03.

```
load 'atx03.test.load(attempt)'
```

or

```
load test(attempt)
```

However, do not specify the following because this results in a search for ATX03.TEST.load.load:

```
load test.load(attempt)
```

Example 3

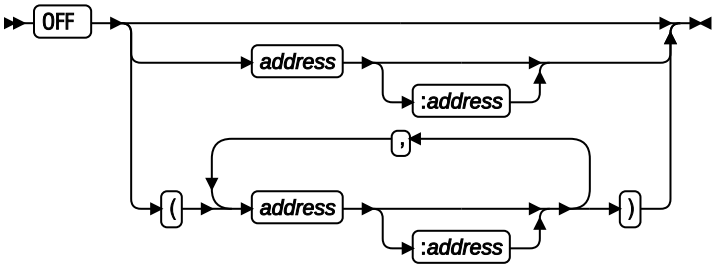
Operation: Load a module named PERFORM from data set ATX03.TRY.

```
load 'atx03.try(perform)'
```

TEST—OFF subcommand

Use the OFF subcommand to remove breakpoints from a program.

TEST—OFF subcommand syntax



TEST—OFF subcommand operands

address
specifies the location of a breakpoint that you want to remove. The address must be on a halfword boundary. If no address is specified, all breakpoints are removed. You can specify *address* as:

- An absolute address
- A symbolic address
- A relative address
- An indirect address
- An address expression
- A module name and entry name (separated by a period)
- An entry name (preceded by a period).

address*:*address
specifies a range of addresses. All breakpoints in the range of addresses are removed. See the description of address for a list of valid address types.

(*address*[,*address*[,*address*[,...]]])
specifies the location of several breakpoints that you want to remove. See the description of address for a list of valid address types.

Note: The list *must* be in parentheses with each address separated by one or more blanks or a comma.

TEST—OFF subcommand examples

Example 1

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

Known:

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

```
off loopc:exitc
```

Example 2

Operation: Remove several breakpoints located at different positions.

Known:

- The addresses of the breakpoints: COUNTRA +2c 3r%

```
off (countra +2c 3r%)
```

Example 3

Operation: Remove all breakpoints in a program.

```
off
```

Example 4

Operation: Remove one (1) breakpoint.

Known:

- The address of the breakpoint is in register 6.

```
off 6r%
```

TEST—OR subcommand

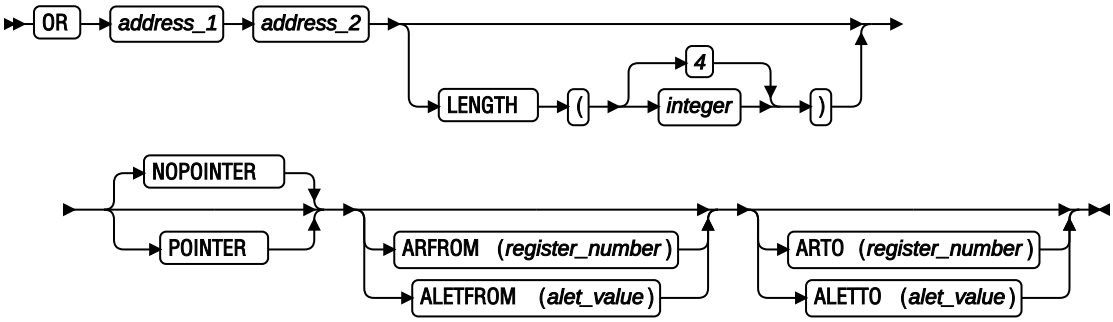
Use the OR subcommand to:

- Alter the contents of the general registers
- OR an entire data field with another

The OR subcommand performs logical OR data or addresses from:

- One virtual storage address to another
- One general register to another
- A general register to virtual storage
- Virtual storage to a general register
- An access register to virtual storage
- Virtual storage to an access register
- One access register to another

TEST—OR subcommand syntax



TEST—OR subcommand operands

address_1
specifies the location of data that is to be ORed with data pointed to by *address_2*.

If you do not specify POINTER and there is a breakpoint in the data pointed to by *address_1*, the TSO/E TEST command processor terminates the OR operation.

address_2
specifies the location of the data that is to be ORed with data pointed to by *address_1*. When the OR operation is complete, the result is stored at this location.

You can specify *address_1* and *address_2* as:

- An absolute address
- A symbolic address
- A relative address
- An indirect address
- An address expression
- A module name and entry name (separated by a period)
- A general register
- An entry name (preceded by a period)
- An access register.

ARTO(*register_number*)
specifies that the location of the data pointed to by *address_2* is in an alternate address/data space referred to by an access register. Valid access register numbers are 0 through 15. The operands ARTO and ALETTO (ALTO) are mutually exclusive.

ARFROM(*register_number*)
specifies that the location of the data pointed to by *address_1* is in an alternate address/data space referred to by an access register. Valid access register numbers are 0 through 15. The operands ARFROM, ALETFROM, and POINTER are mutually exclusive.

ALETTO(*alet_value*) | ALTO(*alet_value*)
specifies that the location of the data pointed to by *address_2* is in an alternate address/data space. The ALETTO value may be from 1 to 8 hexadecimal characters. The operands ALETTO and ARTO are mutually exclusive.

ALETFROM(*alet_value*) | ALFROM(*alet_value*)
specifies that the location of the data pointed to by *address_1* is in an alternate address/data space. The ALETFROM value may be from 1 to 8 hexadecimal characters. The operands ALETFROM, ARFROM, and POINTER are mutually exclusive.

LENGTH(*integer*) | LENGTH(4)
specifies the length, in decimal, of the field to be copied. If an integer is not specified, LENGTH defaults to 4 bytes. The maximum length is 256 bytes.

POINTER
specifies *address_1* is to be validity checked to see that it does not exceed maximum virtual storage size. *address_1* is then 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). When using the POINTER operand, do not specify a general register as *address_1*. The operands ARFROM, ALETFROM, and POINTER are mutually exclusive.

NOPOINTER
specifies *address_1* is to be treated as an address. If neither POINTER nor NOPOINTER is specified, NOPOINTER is the default.

The OR subcommand treats the 16 general registers as contiguous fields. You can OR 10 bytes from general register 0 to another location as follows:

```
or 0r 80060. length(10)
```

The OR subcommand ORs the 4 bytes of register 0, the 4 bytes of register 1, and the high-order 2 bytes of register 2 to virtual storage beginning at location 80060. When a register is specified as *address_1*, the maximum length of data that is ORed is the total length of the general registers or 64 bytes.

TEST—OR subcommand examples

Example 1

Operation: OR two fullwords of data, each in a virtual storage location, placing the result in the second location.

Known:

- The starting address of the data: 80680
- The starting address of where the data is to be: 80690

```
or 80680. 80690. length(8)
```

Example 2

Operation: OR the contents of the two registers, placing the result in the second register specified.

Known:

- The register which contains data specified as the first operand: 10
- The register which contains data specified as the second operand and the result: 5

```
or 10r 5r
```

Example 3

Operation: Turn on the high-order bit of a register.

Known:

- The OR value: X'80'
- The register: 1

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

Note: Specifying the pointer operand causes 80 to be treated as an immediate operand and not as an address.

Example 4

Operation: OR 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 ORed: 14
- The virtual storage location which contains the second operand and result: 80680
- The length of the data to be ORed: 8 bytes

```
or 14r% 80680. l(8)
```

Example 5

Operation: General register 1 points to data in the address/data space referred to by access register 1. OR four bytes where general register 1 points into location A080 in the address/data space referred to by the ALET 40C3A.

```
or 1r? A080. arfrom(1) aletto(40c3a)
```

TEST—PROFILE command

Use the PROFILE command to establish, change, or list your user profile. For a description of the PROFILE command syntax and function, see the “PROFILE command” on page 228.

TEST—PROTECT command

Use the PROTECT command to control unauthorized access to a non-VSAM data set. For a description of the PROTECT command syntax and function, see the “PROTECT command” on page 235.

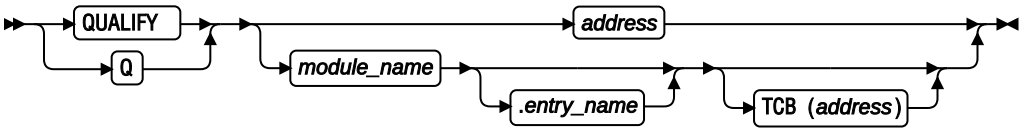
TEST—QUALIFY subcommand

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 uniquely specify which program and which CSECT within that program you intend to test using symbolic and relative addresses.

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

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

TEST—QUALIFY subcommand syntax



TEST—QUALIFY subcommand operands

address
specifies the base location to be used in determining the absolute address for relative addresses only. It does not affect symbolic addressing. You can specify *address* as:

- An absolute address
- A symbolic address
- A relative address
- An indirect address
- An address expression
- A module name and entry name (separated by a period)
- An entry name (preceded by a period).

***module_name*[*.entry_name*]**
specifies the name by which a load module is known, and optionally, an externally referable name within a module. If only a module is specified, the main entry point in the module will be supplied.

TCB(*address*)
specifies the address of a task control block (TCB). This operand is necessary when programs of the same name are assigned to two or more subtasks and you must establish uniquely which one is to be qualified.

Note: When using QUALIFY in combination with other subcommands of TEST (with relative addressing) for routines such as user exit routines, validity check routines, and subtasking, the load module or CSECT indicated might differ from the one that was qualified. This is due to system control processing of automatic qualification.

TEST—QUALIFY subcommand examples

Example 1

Operation: Establish the absolute address 5F820 as a base location for relative addressing.

```
qualify 5f820.
```

Note: This is useful in referring to relative addresses (offsets) within a control block or data area.

Example 2

Operation: Establish a base location for resolving relative addresses.

Known:

- The module name is BILLS.

```
qualify bills
```

Example 3

Operation: Establish an address as a base location for resolving relative addresses.

Known:

- The address is 8 bytes past the address in register 7.

```
q 7r%+8
```

Example 4

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

Known:

- The base address: QSTART

```
qualify qstart
```

Example 5

Operation: Establish a symbol as a base location for resolving relative addresses.

Known:

- The module name: MEMBERS
- The CSECT name: BILLS
- The symbol: NAMES

```
qualify members.bills.names
```

Example 6

Operation: Define the base location for relative and symbolic addressing.

Known:

- The base location is the address of a program named OUTPUT.

```
q output
```

Example 7

Operation: Change the currently qualified module and CSECT. This means defining the base location for relative and symbolic addresses to a new program. The module can be a unique name under any task, or a module under the current task. If there is another one by the same name under a different task, the module under the current task would be used.

Known:

- The module name: PROFITS
- The CSECT name: SALES

```
qualify profits.sales
```

Example 8

Operation: Change the base location for symbolic and relative addresses to a module that has the same name as another module under a different task.

Known:

- The module name: SALESRPT
- The specified module is the one under the task represented by the TCB whose address is in general register 5.

```
q salesrpt tcb(5r%)
```

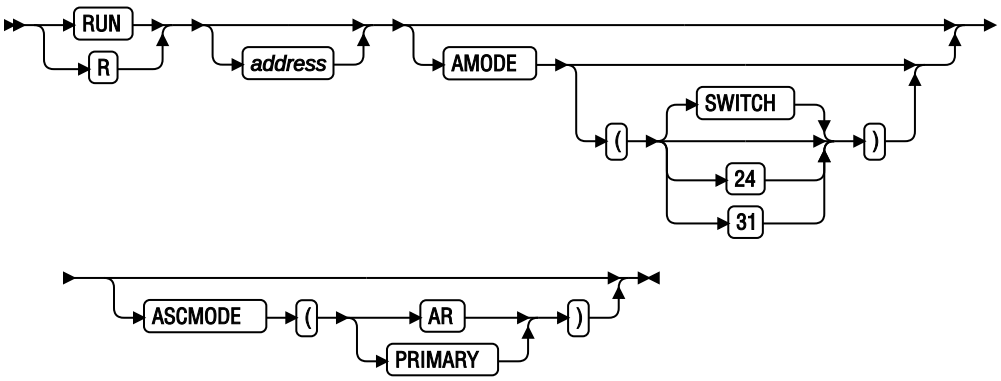
TEST—RENAME command

Use the RENAME command to change the name of a non-VSAM cataloged data set or a member of a PDS, or to create an alias for a member of a partitioned data set. For a description of the RENAME command syntax and function, see the “RENAME command” on page 249.

TEST—RUN subcommand

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.

TEST—RUN subcommand syntax



TEST—RUN subcommand operands

address
execution begins at the specified address. If you do not specify an address, execution begins at the last point of interruption or at the entry point, if the GO or CALL subcommand was not previously specified. You can specify *address* as:

- An absolute address
- A symbolic address
- A relative address
- An indirect address
- An address expression
- A module name and entry name (separated by a period)
- An entry name (preceded by a period).

AMODE [(24 | 31 | SWITCH)]
specifies the addressing mode in which program execution resumes after the RUN subcommand has been issued. You can specify AMODE with RUN, even if the address is not given. However, if AMODE or any abbreviation of AMODE is defined as a symbolic address, it should not be specified with RUN if your intention is to start execution at the address pointed to by AMODE. If RUN AMODE is specified, program execution starts at the last breakpoint and the SWITCH default is taken. If AMODE(SWITCH) is specified, program execution resumes in the addressing mode, which was non-current when RUN was issued. The current addressing mode can be determined by issuing the LISTPSW command.

Note the following:

- If you do not specify AMODE, there is no change in addressing mode.
- If you specify RUN with no operands, the program being tested is restarted at the next executable instruction. However, if the tested program has abended in an address space other than home, the home and primary address space identifiers (ASIDs) are different, and the instruction address in the PSW refers to an address space which TEST cannot access. Therefore, do not specify RUN without operands after such an abend.

ASCMODE(AR | PRIMARY)
specifies the PSW mode in which program execution resumes after you issue the RUN subcommand. If you specify ASCMODE(PRIMARY), the PSW mode is set to execute the program using the primary address space control mode (in primary mode). When ASCMODE(AR) is specified, the PSW is set to execute the program in AR mode.

TEST—RUN subcommand examples

Example 1

Operation: Execute a program to termination from the most recent point of interruption.

```
run
```

Example 2

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

Known:

- The address: +A8

```
run +a8
```

TEST—SEND command

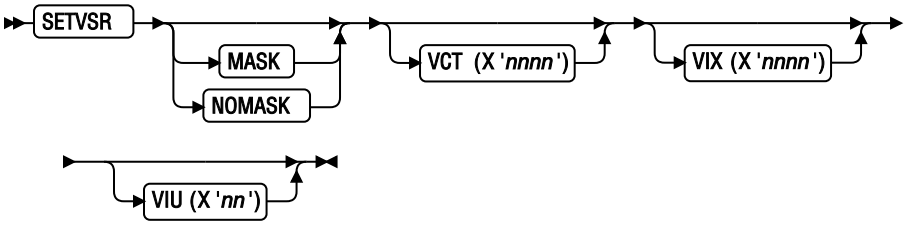
Use the SEND command to send a message to another terminal user or to the system operator. For a description of the SEND command syntax and function, see the “SEND command” on page 254.

TEST—SETVSR subcommand

Use the SETVSR subcommand to set fields in the vector status register. The SETVSR subcommand allows you to:

- Specify the vector mask register control mode
- Update the vector count
- Update the vector interruption index
- Update the vector in-use bits.

TEST—SETVSR subcommand syntax



TEST—SETVSR subcommand operands

MASK | NOMASK
specifies the vector mask register control mode.

VCT(X'nnnn')
allows you to update the vector count. (X'nnnn') specifies the number of vector elements that are to be processed.

VIX(X'nnnn')
allows you to update the vector interruption index. (X'nnnn') specifies the vector element that processing is to start with.

VIU(X'nn')
allows you to update the vector in-use bits. (X'nn') specifies active register pairs.

TEST—SETVSR subcommand examples

Example 1

Operation: Set values in the vector status register.

Known:

- The vector mask register control mode is to be **NOMASK**
- The vector count in hexadecimal: 75
- The vector interruption index in hexadecimal: 76
- The vector in-use bits in hexadecimal: B1

```
setvsr nomask vct(x'75') vix(x'76') viu(x'b1')
```

TEST—STATUS command

Use the STATUS command to display the status of batch jobs at the terminal. For a description of the STATUS command syntax and function, see the "STATUS command" on page 265.

TEST—SUBMIT command

Use the SUBMIT command to submit one or more batch jobs for processing under TEST. For a description of the SUBMIT command syntax and function, see the "SUBMIT command" on page 266.

TEST—TERMINAL command

Use the TERMINAL command to define the operating characteristics for the type of terminal you are using. For a description of the TERMINAL command syntax and function, see the "TERMINAL command" on page 270.

TEST—UNALLOC command

Use the UNALLOC command to release (deallocate) previously allocated data sets that are no longer needed. Because FREE is an alias of the FREEMAIN subcommand, use UNALLOC to free files under TEST. For a description of the FREE command syntax and function, see the "FREE command" on page 138.

TEST—WHERE subcommand

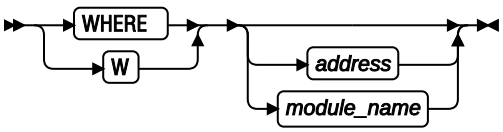
Use the WHERE subcommand to obtain:

- An absolute address
- The name of a module and CSECT
- A relative offset within the CSECT
- The address of the TCB for the specified address.

You can also use the WHERE subcommand to obtain the absolute address serving as the starting or base location for the symbolic and relative addresses in the program. Alternately, you can obtain the absolute address of an entry point in a particular module or control section (CSECT). If you do not specify any operands for the WHERE subcommand, you receive the address of the next executable instruction, the related load module and CSECT names, and the hexadecimal offset.

Note: After an abend outside the home address space, do not specify WHERE without operands. The home and primary address space identifiers (ASIDs) are different after an abend, resulting in an instruction address which TEST cannot access.

TEST—WHERE subcommand syntax



TEST—WHERE subcommand operands

address

You can specify *address* as:

- An absolute address

- A symbolic address
- A relative address
- An indirect address
- An address expression
- A module name and entry name (separated by a period)
- An entry name (preceded by a period).

If you specify WHERE without an address, the address of the next executable instruction, the related load module and CSECT names, and the hexadecimal offset are displayed.

module_name
specifies the name by which a load module is known or the name of an object module. The output of the WHERE subcommand is the module name, the CSECT name, the offset within the CSECT, the absolute address, and the address of the TCB. If only the module name was specified, the only output is the absolute address of the module and the address of the TCB for the task under which the module is found.

If the specified address is *not* within the extent of any user program, only the absolute address is returned. Along with the absolute address, a message will be returned stating that the specified address is not within the program extent. If no operands are specified, the absolute address returned is the address of the next executable instruction.

TEST—WHERE subcommand examples

Example 1

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

```
where
```

Example 2

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

Known:

- The absolute address: 3E2B8

```
where 3e2b8.
```

Example 3

Operation: Obtain absolute address of +2c4.

```
w +2c4
```

Note: An unqualified relative address is calculated from the currently qualified address (as specified using the QUALIFY command or the current module and CSECT, if no other qualification exists). The module name, CSECT name, and TCB address are also obtained along with the absolute address.

Example 4

Operation: Obtain offset of the symbol SALES in the current program.

```
where sales
```

Note: The module name, CSECT name, absolute address, and the TCB address are returned along with the offset of SALES.

Example 5

Operation: Determine in which module the address in register 7 is located.

```
w 7r%
```

Note: The offset, absolute address, and the TCB address are also returned with the module name.

Example 6

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

```
where cstart
```

Example 7

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

```
where netsales.july
```

Example 8

Operation: Determine the relative address of symbol COMPARE in the module named CALCULAT and CSECT named AVERAGE.

```
w calculat.average.compare
```

Note: The absolute address and TCB address are also returned with the relative address.

Example 9

Operation: Determine the virtual address of +1CA.

Known:

- The CSECT: MARCH
- The module: GETDATA

```
where getdata.march.+1ca
```

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

Example 10

Operation: Obtain the absolute address for relative address +2C in CSECT named PRINTIT within the currently qualified module.

```
where .printit.+2C
```

TIME command

Use the TIME command to obtain the following information:

- Cumulative CPU time (from LOGON)
- Cumulative session time (from LOGON)
- Total service units used, which includes:
  - CPU service units - A measure of task execution time.
  - I/O service units - A measure of SMF data set activity.
  - Storage service units - A measure of the page frame usage.
- Local time of day

Refers to the time of execution for this command. It is displayed as follows:

```
local time of day in hours(HH),
minutes(MM), and seconds(SS),
(am or pm is also displayed)
```

- Today's date.

To enter the command while a program is executing, you must first cause an attention interruption. The TIME command has no effect on the executing program.

TIME command syntax

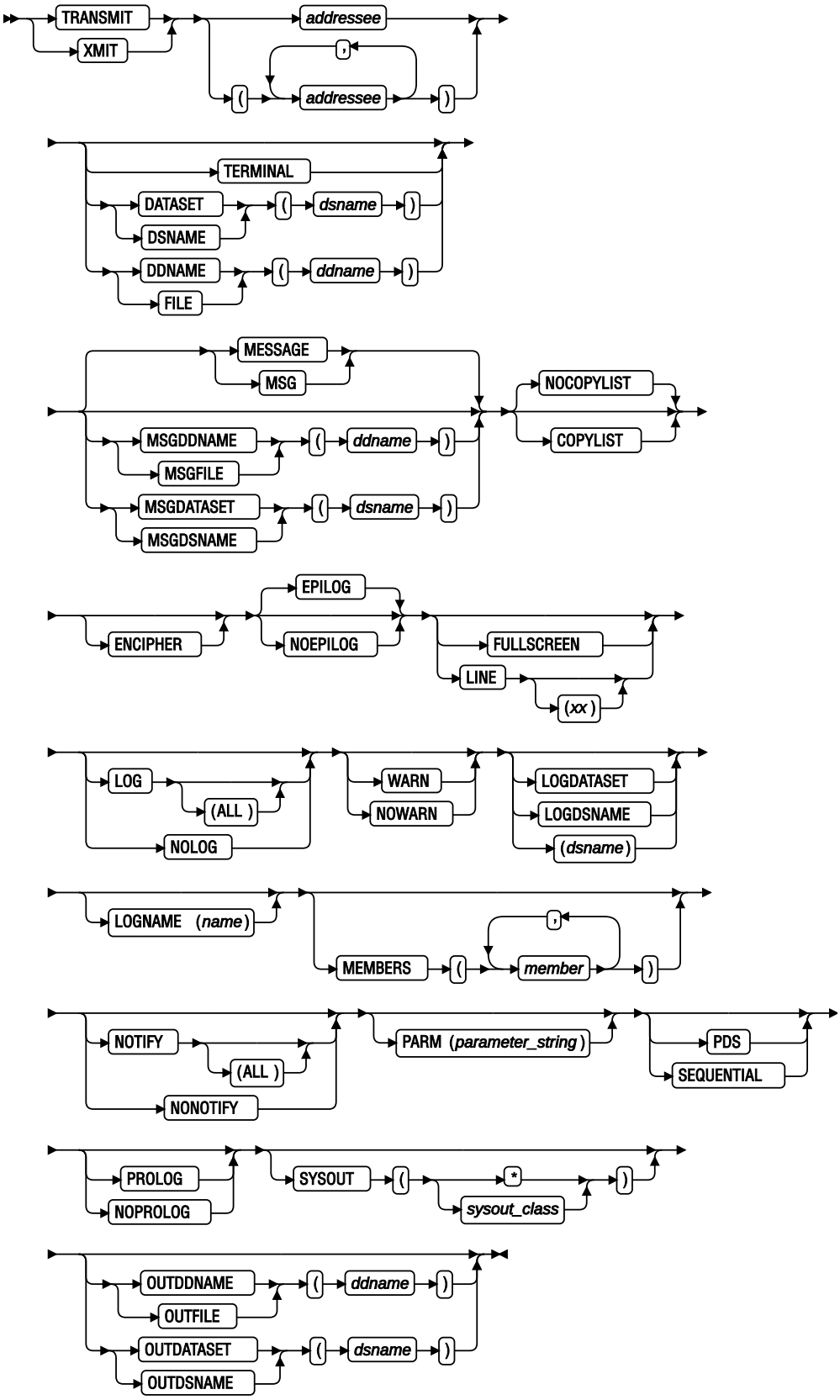
TIME command return code

The return code is always zero.

TRANSMIT command

Use the TRANSMIT command to send information (a message), or a copy of information (a data set), or both, to another user. The TRANSMIT command converts this data into a special format so that it can be transmitted to other users in the network. Use the RECEIVE command to retrieve the data and restore it to its original format. TRANSMIT is an APF authorized command, and therefore it cannot be called from an unauthorized program. See “TSOEXEC command” on page 347.

TRANSMIT command syntax



TRANSMIT command operands

Note: If you specify either MSGDDNAME or MSGFILE, or MSGDATASET or MSGDSNAME, the TERMINAL operand is no longer the default.

(*addressee*[,*addressee*[, ...]])
specifies the information identifying the target user(s). You can combine one or more of the following: a node and user ID specified as *node.user_id* or *node/user_id*, a nickname, or a distribution list name. If you identify only one user as the addressee, you can omit the parentheses. See "NAMES data set function" on page 342.

A maximum of 200 node/userid combinations and 200 nicknames may be specified, but the total number of addressees may not exceed 200.

DATASET(*dsname*) | DSNAME(*dsname*)
specifies the name of a data set to be transmitted. The data set must be on a direct access storage device (DASD).

DDNAME(*ddname*) | FILE(*ddname*)
specifies the 1- to 8-character ddname of a preallocated file to be transmitted. The data set must be on a direct access storage device (DASD). For concatenations, the first data set cannot be empty. If you transmit a member of a preallocated partitioned data set, you must specify the MEMBERS operand.

TERMINAL
specifies data input is to be taken from the terminal. You are prompted to enter data to be transmitted either in line mode or in full-screen mode as specified by the LINE or FULLSCREEN operand.

MSGDDNAME(*ddname*) | MSGFILE(*ddname*)
specifies a 1 to 8 character ddname or file name of the file that is to be transmitted. You must allocate the file before it is transmitted. The system transmits the file as a message.

The file must have a record format of either FB or F and a record length of 80. You can specify a sequential data set or a member of a partitioned data set. MSGDDNAME or MSGFILE is mutually exclusive with MSGDATASET or MSGDSNAME, and MESSAGE or MSG.

If you specify either MSGDDNAME or MSGFILE, the TERMINAL operand is no longer the default. This allows you to send the data or message to be displayed at the recipient's terminal without having to enter the data or message either in line mode or in full-screen mode. If you want full-screen mode, you must explicitly specify TERMINAL.

When you specify the ENCIPHER operand, the following can happen:

- If you specify the ENCIPHER operand and either the TERMINAL, DATASET, DDNAME, DSNAME, or FILE operands, the system does *not* encipher the data set specified with the MSGDDNAME or MSGFILE operand.
- If you specify ENCIPHER and do *not* specify the TERMINAL, DATASET, DDNAME, DSNAME, or FILE operands, the system enciphers the data set specified with the MSGDDNAME or MSGFILE operand.

The ENCIPHER operand is described later in this section.

MSGDATASET(*dsname*) | MSGDSNAME(*dsname*)
specifies the data set that is to be transmitted. The system transmits the data set as a message.

The data set must have a record format of either FB or F and a record length of 80. You can specify a sequential data set or a member of a partitioned data set. MSGDATASET or MSGDSNAME is mutually exclusive with MSGDDNAME or MSGFILE, and MESSAGE or MSG.

If you specify either MSGDATASET or MSGDSNAME, the TERMINAL operand is no longer the default. This allows you to send the data or message to be displayed at the recipient's terminal without having to enter the data or message either in line mode or in full-screen mode. If you want full-screen mode, you must explicitly specify TERMINAL.

When you specify the ENCIPHER operand, the following can happen:

- If you specify the ENCIPHER operand and either the TERMINAL, DATASET, DDNAME, DSNAME, or FILE operands, the system does *not* encipher the data set specified with the MSGDATASET or MSGDSNAME operand.
- If you specify ENCIPHER and do *not* specify the TERMINAL, DATASET, DDNAME, DSNAME, or FILE operands, the system enciphers the data set specified with the MSGDATASET or MSGDSNAME operand.

The ENCIPHER operand is described later in this section.

MESSAGE | MSG
specifies that you are to be prompted for messages that accompany a transmitted data set. The prompt is either in full-screen mode or in line mode, depending on the terminal type and the specification of FULLSCREEN or LINE.

Note the following:

- If you specify both TERMINAL and MESSAGE, TSO/E prompts you twice for the data.
- TSO/E uses the prefix as the high-level qualifier for the name of the data set to be transmitted.

COPYLIST | NOCOPYLIST

COPYLIST
specifies that TRANSMIT build a list of the specified addressees and append it as a prolog to the message. If a data set is being transmitted, the copylist is added as an accompanying message. If a message is being transmitted, COPYLIST prefixes the message text.

NOCOPYLIST
specifies no copylist is to be generated or appended. NOCOPYLIST is the default.

ENCIPHER
specifies TRANSMIT should encipher the data by invoking the Access Method Services REPRO command. The TRANSMIT command prompts for ENCIPHER options to be passed with the REPRO command.

EPILOG | NOEPILOG

EPILOG
specifies TRANSMIT should include epilog lines from the NAMES data set, if a terminal message is transmitted. An EPILOG is added unless you either type in NOEPILOG or have no epilog in your NAMES data set. EPILOG is the default.

NOEPILOG
specifies no epilog lines should be included.

FULLSCREEN | LINE | LINE(*nn*)

FULLSCREEN
requests all terminal input for messages or data be read in full-screen mode. This is the default for 3270 terminals capable of supporting a minimum screen size of 24 rows by 80 columns.

LINE | LINE(*nn*)
requests terminal input for messages and data be read in single-line mode. This is the default for non-3270 terminals. Use *nn* in a 1 to 2 character string to mark the end of data. You can also use LINE(*nn*) to allow a CLIST to provide messages or data. To terminate message input, enter a null line or the 1 or 2 character string value LINE(*nn*) in columns 1 and 2. LINE(*nn*) allows you to insert blank lines into the text. Leading blanks are eliminated when in a CLIST, but they are kept when not in a CLIST.

LOG | NOLOG | LOG(ALL)

LOG
records the transmission in the LOG data set. LOG does not necessarily indicate that the log entry contains a line for every addressee except for *node.userid* addressees. The LOG/NOLOG/LOGLST tags in the nicknames section of the NAMES data set or the LOG/NOLOG tags in the control section of the NAMES data set determine whether the log entry contains addressee entries for a nickname or distribution list. Only one log entry is built in the default log file per transaction. LOG is the

default. See "Logging function of TRANSMIT and RECEIVE" on page 341. To ensure that the log entry contains a line for each addressee, including those on a distribution list, specify the LOG(ALL) option. See LOG(ALL) for more information.

NOLOG
specifies not to record the transmission in the LOG data set. NOLOG overrides all LOG/LOGLST tags in the NAMES data set.

LOG(ALL)
specifies the log entry contain a line for each addressee, including those derived from any distribution lists on the NAMES data set. This specification overrides the NOLOG/NOLOGLST tags in the NAMES data set.

LOGDATASET(*dsname*) | LOGDSNAME(*dsname*)
specifies an alternate name of a sequential data set in which to log the transmitted data. Users who are defined to more than one security label might need to specify a log data set name if they are logged on at a security label other than the SECLABEL of the profile that is protecting their log data set. A user's current security label (the security label the user is logged on with) must match the security label of the log data set in order for a transmission to be logged in the data set. Specifying a log data set allows users to log transmissions for each security label they are defined to in separate data sets. The data set must have a logical record length of 255, a record format of variable blocked, and a block size of 3120. If the data set does not exist, the system creates it.

LOGNAME(*name*)
uses the name as the LOGNAME qualifier on the log data set name. See "Logging function of TRANSMIT and RECEIVE" on page 341.

MEMBERS(*member*)
transmits a list of members from the specified partitioned data set.

NOTIFY
notifies the sender when the data has been received. NOTIFY does not necessarily guarantee that notification is requested except for *node*.*userid* addressees. For nicknames and distribution lists, control of notification is determined by the :NOTIFY or :NONOTIFY tag in the nickname section of the NAMES data set.

NOTIFY(ALL)
notifies the sender when the data has been received by all addressees. This operand overrides the :NOTIFY or :NONOTIFY tags in the nickname entries of the NAMES data set or distribution lists.

NONOTIFY
suppresses the notify function. This stops the notify function completely, overriding any specification in the NAMES data set or in the distribution lists.

PARM(*parameter_string*)
Your installation may instruct you to use this operand to specify installation-dependent data.

PDS | SEQUENTIAL

PDS
unloads a member or members of a partitioned data set (PDS) before transmission. This method preserves the directory information, but forces the receiving user to restore the member(s) into a PDS. PDS is the default.

Note: Some non-MVS systems cannot receive a partitioned data set. To transmit a member of a partitioned data set or a sequential data set, use the SEQUENTIAL keyword. For more information about data set transmission, see the SEQUENTIAL keyword description.

SEQUENTIAL
sends a member of a partitioned data set or a sequential data set as a sequential data set. This method does not preserve directory information, but allows the receiving user to restore the data set as either a sequential data set or as a member of a partitioned data set. If transmission is by ddname, the member must be preallocated. The SEQUENTIAL keyword is ignored when no member is specified for a partitioned data set.

PROLOG | NOPROLOG

PROLOG
specifies TRANSMIT should include prolog lines from the control section of the NAMES data set, if a terminal message is transmitted. PROLOG is the default.

NOPROLOG
specifies not to include prolog lines.

SYSOUT(*sysoutclass* | *)
uses the SYSOUT class for messages from utility programs, which are used by TRANSMIT (for example IEBCOPY). If you specify a * (asterisk), TSO/E directs utility program messages to the terminal. The default is typically *, but the installation can modify it.

OUTDDNAME(*ddname*) | OUTFILE(*ddname*)
specifies the use of a preallocated file as the output data set for the TRANSMIT command. No data is written to SYSOUT for transmission and the system limit on the number of records that can be transmitted does not apply. TSO/E assigns the DCB attributes as LRECL=80, BLKSIZE=3120, and RECFM=FB. Specify the ddname as either a sequential data set or a member of a partitioned data set.

Use OUTDDNAME or OUTFILE with the INDDNAME or INFILE operand of the RECEIVE command. OUTDDNAME and OUTFILE are primarily intended for system programmer use.

OUTDSNAME(*dsname*) | OUTDATASET(*dsname*)
specifies the use of a data set as the output data set for the TRANSMIT command. No data is written to SYSOUT for transmission and the system limit on the number of records that can be transmitted does not apply. TSO/E assigns the DCB attributes as LRECL=80, BLKSIZE=3120, and RECFM=FB. The data set must be a sequential data set.

Use OUTDSNAME or OUTDATASET with the INDSNAME or INDATASET operand of the RECEIVE command. OUTDSNAME and OUTDATASET are primarily intended for system programmer use.

WARN | NOWARN

WARN
You can request that the TRANSMIT command issues warning message INMX034I when the warning threshold is initially met, and thereafter whenever the warning interval is met. (This is the default if WARN or NOWARN is specified.)

NOWARN
You can request that the TRANSMIT command does not issue warning message INMX034I when the warning threshold is initially met, nor thereafter whenever the warning interval is met.

TRANSMIT command return codes

Table 56 on page 339 lists all the return codes of TRANSMIT command.

Table 56: TRANSMIT command return codes

Return code	Meaning
0	Processing successful.
4	Processing successful, but a warning message has been issued.
8	Processing incomplete. At least one transmission was unsuccessful.
12	Processing unsuccessful.
16	Processing unsuccessful. Abnormal end.

Transmitting data sets

You can use the TRANSMIT command to transmit sequential or partitioned data sets with record formats of F, FS, FB, FBS, V, VB, VBS, and U. The data sets must reside on a direct access storage device (DASD). For a VB or VBS data set, the largest logical record length (LRECL) TSO/E can transmit to VM is 65,535. Data sets with machine and ANSI print-control characters are also supported. TRANSMIT does not support data sets with the following types of values:

- keys
- ISAM data sets
- VSAM data sets
- data sets with user labels

If a partitioned data set (PDS or PDSE) is transmitted, it is unloaded with IEBCOPY and then the unloaded version is transmitted. If a single member is transmitted, it is generally unloaded before transmission. You can force transmission of a partitioned data set member as a sequential data set by using the SEQUENTIAL operand. Forced transmission of a partitioned data set member as a sequential data set does not preserve the directory information. The IEBCOPY unload preserves directory information, but the receiver must reload it into a partitioned data set.

Transmitting data sets as messages

You can transmit a data set as a message by specifying MSGDDNAME or MSGFILE, or MSGDATASET or MSGDSNAME. Using these operands might reduce the time it takes to transmit a file or data set. The file or data set must have a record format of fixed block (FB) and a record length (LRECL) of 80. You can transmit either a sequential data set or a member of partitioned data set. For MSGDDNAME or MSGFILE, you must allocate the file before you transmit.

Transmitting messages

If you specify MESSAGE when you transmit data, TRANSMIT prompts you for messages that accompany the data. These messages are shown to the receiving user when the RECEIVE command is issued. The system displays these messages to the receiving user when the RECEIVE command is issued and before the user is prompted to perform some action about the data.

You can enter up to 220 lines of data in either full-screen mode or single line mode. Of the 220 lines of data, ten are reserved for the PROLOG lines. If you specify the EPILOG tag in the NAMES data set, you can specify an additional 10 lines beyond the 220 line limit. For full-screen mode, use the program function (PF) keys for scrolling (PF7 or PF19 and PF8 or PF20) and for termination (PF3 or PF15). For single line mode, messages are terminated by either a null line or the string value specified in LINE(*nn*).

Note: Full-screen mode is the default for 3270 terminals capable of supporting a minimum screen size of 24 rows by 80 columns.

Transmitting a message that you enter from the terminal is the simplest form of the TRANSMIT command. You specify TRANSMIT addressee-list and TRANSMIT defaults to terminal input. Messages sent in this manner are not saved in a data set, but are saved in the LOG data set.

Transmitting enciphered data

To encipher the transmitted data, specify the ENCIPHER operand. The TRANSMIT command prompts for encipher options, which are passed to the access method services REPRO command.

Data encryption function of TRANSMIT and RECEIVE

The TRANSMIT and RECEIVE commands support encryption using DFSMS.

TSO/E uses the access method services REPRO command to encrypt data sets before transmitting them. However, your installation must allow encryption.

If you have either of the programs installed and your installation allows encryption, TRANSMIT, as required, invokes the access method services REPRO command to encrypt data sets before they are

transmitted. The TRANSMIT and RECEIVE commands prompt you for encipher/decipher options and append what you entered as REPRO command suboperands of the ENCIPHER or DECIPHER operand.

Transmitting data sets and messages with security labels

If your installation uses security labels and security checking, any data sets or messages you transmit have a security label associated with them. The security label you are logged on with when you issue the TRANSMIT command is the one associated with the data. In order for receivers to view the data set or message, they must be logged on with a security label that is equal to or greater than the one associated with the data.

Some considerations for transmitting and receiving data sets and messages with security labels are:

- Receivers can only receive data sets and messages that they are authorized to receive based on the security label they are logged on with.
- To receive data sets and messages with a greater security label, receivers can log on with a greater security label if their TSO/E user IDs are authorized to do so. They can then use the RECEIVE command to view the messages and data sets.
- If the receivers cannot log on with a security label that allows them to view the transmitted data (data set or message), the system deletes the data, unless your installation uses a JES installation exit to take some other action.
- The receivers do not receive a notice that they have data sets or messages to receive if the data was transmitted with a greater security label than the receivers are logged on with.

Logging function of TRANSMIT and RECEIVE

The TRANSMIT and RECEIVE functions normally log each file transmitted and received. The TRANSMIT and RECEIVE commands create appropriate log data sets, if they do not already exist.

The name of the log data set is determined as follows:

1. If the LOGDATASET or LOGDSNAME operand is used, the data set ‘prefix.logdsname’ is used for logging.
2. In the absence of any user or installation specification, the default log data set name is ‘prefix.LOG.MISC’.
3. The qualifier LOG is called the log selector and can be changed by the :LOGSEL tag in the control section of the NAMES data set. This qualifier is common for all log data sets belonging to any given user.
4. The qualifier MISC is called the log name. It might be overridden by the LOGNAME operand on the TRANSMIT command, the :LOGNAME tag in the control section of the NAMES data set, or by the :LOGNAME tag in a nickname definition.

Use the log selector to define all of your log data sets under one name. The log name identifies each individual data set in the log data set. For example, you can list all of your log data sets by ‘prefix.LOG’. This would give you a list of all of your log data sets with the individual log names.

The log data sets have the following DCB attributes: LRECL=255, BLKSIZE=3120, and RECFM=VB.

With any given invocation of the TRANSMIT or RECEIVE command, logging can occur to more than one log data set depending upon the presence of the :LOGNAME tag on the nickname or distribution list entry in the NAMES data set. However, with any given invocation of the TRANSMIT or RECEIVE command, only one log entry is written to any one log data set. This log entry then contains an addressee entry for each addressee being logged to that log data set.

The first lines in each log entry contain a line of hyphens and a descriptor line. The format of the descriptor line is:

Column
Usage

1 - 8
Name of the command using the entry.

17 - 60
Name of the data set transmitted or received.

63 - 82
Time stamp from the command execution.

For the TRANSMIT command log entries, subsequent lines indicate the addressees to which the transmission was sent, the names of any members of a partitioned data set selected for transmission, and any messages entered with the TRANSMIT command.

For the RECEIVE command log entries, the second log line always identifies the originator of the transmission. The originator of the transmission can be the issuer of the TRANSMIT command (in the case of a file or message receipt) or the issuer of the RECEIVE command (if the log entry is for notification). If the entry in the log is a file or a message receipt, the time stamp recorded is from the TRANSMIT command. If the entry in the log was a notification, the time stamp is from the RECEIVE command. The format is:

Column
Usage

9 - 15
Nickname of the originating user or blanks.

17 - 24
Node name of the originating user.

26 - 33
User ID of the originating user.

35 - 61
Name of the originating user or blank.

63 - 82
Time stamp from the originating command.

For RECEIVE command notification entries, the third log line identifies the original transmission. The data set name and time stamp on this line are those from the original transmission. The format of the third log line is:

Column
Usage

4 - 15
Error code from RECEIVE. STORED indicates that the RECEIVE operation was successful.

17 - 60
Data set name from the TRANSMIT command.

63 - 82
Time stamp from the TRANSMIT command.

NAMES data set function

The TRANSMIT command allows several different specifications of a list of addressees. The simplest is a single addressee whose node name and user ID are specified explicitly. The next level is the nickname specification. The nickname is a 1 to 8 character name that is a synonym for the node and user ID. The TRANSMIT and RECEIVE commands find the actual node and user ID by looking up the nickname in tables provided in the NAMES data set. The final level of addressing is a distribution list. A definition in the NAMES data set identifies a distribution list name. The named list can reference up to 100 nicknames of either addressees or other distribution lists.

Each user of the TRANSMIT and RECEIVE commands can have one or more NAMES data sets to resolve nicknames and establish the default mode of operation. In the absence of any explicit installation specification, the name of the first of these data sets is ‘prefix.NAMES.TEXT’. The first data set contains the names of any other NAMES data sets. The data set can have either fixed or varying length records.

Using varying length records will save disk space. The records are numbered according to standard TSO/E conventions. They can also be unnumbered. The data set is either blocked or unblocked with any record length less than or equal to 255.

The data set is composed of two sections, the control section and the nicknames section. The control section must precede the nicknames section. The control section ends at the first :NICK tag. Use the control section to set defaults for LOG/NOLOG and NOTIFY/NONOTIFY, prolog or epilog lines, the default log data set name, and to identify other NAMES data sets that are used.

The nicknames section contains one entry for each nickname and distribution list name that you want to define.

Each occurrence of a colon in the NAMES data set is treated as the start of a tag. If the tag following the colon is not one of those described later in this section, it is treated as a user-defined tag that may be processed by an installation-written application that uses the NAMES data set. The information that follows a user-defined tag is ignored by TRANSMIT and RECEIVE processing. For more information about installation-written applications, see *z/OS TSO/E Programming Guide*.

Control section tags

Use the beginning of the NAMES data set to control certain operations of the TRANSMIT and RECEIVE commands. The tags are optional. You can include any of the following tags:

:ALTCTL.*dsname*
specifies the fully-qualified file name of another file to be used in the nickname look up process. If TRANSMIT finds more than one :ALTCTL tag, TRANSMIT uses the order of the :ALTCTL tags to scan the files. You can specify up to ten :ALTCTL tags. All control section tags, the :LOG and :NOLOG tags, the :LOGNAME tag, and the :NOTIFY and :NONOTIFY tags are always ignored when read from any alternate NAMES data set.

:EPILOG.*text*
in the control section, specifies a text line to be appended at the end of any transmitted message. The maximum length of an epilog line is 72 characters. You can specify up to ten :EPILOG lines. If more than one :EPILOG record is found, records appear in the message in the same order as they are in the file. Text data for the :EPILOG tag must be on the same line as the :EPILOG tag.

:PROLOG.*text*
in the control section, specifies a text line to be inserted at the beginning of any transmitted message. The maximum length of a prolog line is 72 characters. You can specify up to ten :PROLOG lines. If more than one :PROLOG record is found, records appear in the message in the same order as they are in the NAMES data set. Text data for the :PROLOG tag must be on the same line as the :PROLOG tag.

:LOGNAME.*name*
in the control section, serves as a default qualifier for the log data set name. If you specify it in the nickname entry, the value provided overrides the default set in the control section. See "Logging function of TRANSMIT and RECEIVE" on page 341.

:LOGSEL.*name*
in the control section, specifies the second (middle) qualifier of all log data sets. See "Logging function of TRANSMIT and RECEIVE" on page 341.

:LOG | :NOLOG
in the control section, indicates whether you want logging for any addressee specified by node and user ID and for any nickname that does not also specify :LOG or :NOLOG. If the nickname entry contains the :LOG or :NOLOG tag, this value overrides any value in the control section. However, it might have been overridden by a specification on the TRANSMIT command. If you specify NOLOG in your NAMES data set in the control section or on a :NICK tag, TSO/E prompts you with a message to receive data set 'PREFIX.MAIL.*USERID*'. TSO/E then stores and places the message in 'myid.MAIL.*USERID*' where *myid* is the receiver of the message and *USERID* is the originator of the message.

The default is :NOLOG.

:NOTIFY | :NONOTIFY
in the control section, indicates whether you want notification for any addressee specified by node and user ID, and for any nickname where the nickname entry does not contain :NOTIFY or :NONOTIFY. The value of :NOTIFY or :NONOTIFY in the NAMES data set might be overridden by a similar specification on the TRANSMIT command. If you want to be notified for addressees on distribution lists, you must specify :NOTIFY on the distribution list in the control data set or specify NOTIFY(ALL).

The default is :NOTIFY.

Nicknames section tags

The nicknames section is composed of tags and their values in the same manner as the control section. The nicknames section is different from the control section in that it is divided by the occurrence of each :NICK tag and continues until the next :NICK tag, which starts the next definition. Use the nickname as either a nickname of a single user or the name of a distribution list. The :NODE and :USERID tags are present when you use the nickname for a user definition. The :LIST and :CC tags, or both are present when you use the nickname for distribution list definition.

Use the log and notify tags, except for :LOGLST and :NOLOGLST, with either a user ID definition or a distribution list definition.

Note the following:

1. Each nickname entry must begin with the :NICK tag and :NICK must be the first non-blank character on the line.
2. You can specify the tags as all uppercase or all lowercase.
3. :NICK.*nickname* and :USERID.*user_id* are required.

:NOTIFY | :NONOTIFY
in the control section, specifies whether you want notification for any addressee specified by node and user ID, and for any nickname where the nickname entry does not contain :NOTIFY or :NONOTIFY. The value of :NOTIFY or :NONOTIFY in the NAMES data set might be overridden by a similar specification on the TRANSMIT command. If you want to be notified for addressees on distribution lists, you must specify :NOTIFY on the distribution list in the control data set or specify NOTIFY(ALL).

:NICK.*name*
indicates a nickname entry in the NAMES data set. It must be the first non-blank (except for line numbers) character of the record. The nickname is a 1 to 8 character string of non-blank alphanumeric characters.

:NODE.*node_id*
in the nickname entry, specifies a network node name for the nickname entry. If the :NODE tag is not present in a nickname entry, the local user's node name is assumed.

:USERID.*user_id*
specifies the user ID of the user to be identified by the nickname. You cannot use the :USERID tag with :LIST or :CC tags in the same nickname entry.

:LOG | :NOLOG
in the control section, indicates whether you want logging for any addressee specified by node and user ID and for any nickname that does not also specify :LOG or :NOLOG. If the nickname entry contains the :LOG or :NOLOG tag, this value overrides any value in the control section. However, it might have been overridden by a specification on the TRANSMIT command. If you specify NOLOG in your NAMES data set in the control section or on a :NICK tag, TSO/E prompts you with a message to receive data set ‘A.MAIL.USERID’. TSO/E then stores and places the message in ‘myid.MAIL.USERID’ where *myid* is the receiver of the message and USERID is the originator of the message.

:LOGLST | :NOLOGLST
in the nickname entry, defines a distribution list. The tag indicates whether a log entry should be made for each addressee in the list.

:NAME.*user_name*
specifies the plain text name of the user being defined. This name appears in the copy list and in any log entries for this nickname. You can specify up to 30 characters.

:ADDR.*address*
in the nickname entry, specifies the mailing address of the specified user. Separate individual lines of the address with semicolons.

:LIST.{*name* | *name_list*}
in the nickname entry, specifies a list of addressees that make up the distribution list. Specify the addressee as either a nickname of the name or another distribution list. The :LIST tag can reference up to 100 nicknames. If you want to be notified for addressees on distribution lists, specify :NOTIFY on the distribution list in the control data set or specify NOTIFY(ALL) on the TRANSMIT command.

:CC.{*name* | *name_list*}
specifies further nicknames of addressees for a distribution list. It is treated as a synonym of the :LIST tag. You can specify up to 100 nicknames.

:PARM.*text*
specifies up to 30 characters of installation-defined data. TSO/E passes this data to the RECEIVE command installation exits. For more information about how an installation uses these exits, see *z/OS TSO/E Customization*.

TRANSMIT command examples

In the following examples, the transmitting user is assumed to have user ID USER1 on node NODEA and the receiving user is assumed to have user ID USER2 on node NODEB. The sending user has a NAMES data set as follows:

```
* Control section
:altctl.DEPT.TRANSMIT.CNTL
:prolog.Greetings from John Doe.
:prolog.
:epilog.
:epilog.Yours,:epilog.John Doe :epilog.NODEA.USER1
*
* Nicknames section.
*
:nick.alamo  :list.Jim Davy   :logname.alamo   :notify.
:nick.addrchg :list.joe davy jim  :nolog  :nonotify
:nick.Joe  :node.nodeb  :userid.user2    :name.Joe Doe
:nick.Me   :node.nodea  :userid.user1    :name.me
:nick.Davy :node.alamo  :userid.CROCKETT :name.Davy Crockett
:nick.Jim  :node.ALAMO  :userid.Bowie    :name.Jim Bowie
```

In the examples involving the RECEIVE command, data entered by the user appears in lowercase and data displayed by the system is in uppercase.

Example 1

Transmit a copy of the ‘SYS1.PARMLIB’ data set to Joe, identifying Joe by his node and user ID.

```
transmit nodeb.user2 da('sys1.parmlib')
```

Example 2

Joe receives the copy of ‘SYS1.PARMLIB’ transmitted above.

```
receive
Dataset SYS1.PARMLIB from USER1 on NODEA
Enter restore parameters or 'DELETE' or 'END' +
<null line>
Restore successful to dataset 'USER2.PARMLIB'
---------------------------------------------
No more files remain for the RECEIVE command to process.
```

In the preceding example, Joe has issued the RECEIVE command, seen the identification of what arrived, and chosen to accept the default data set name for the arriving file. The default name is the original data set name with the high-level qualifier replaced by his user ID.

Example 3

Transmit two members of 'SYS1.PARMLIB' to Joe, and add a message identifying what was sent. Joe is identified by his NICKNAME, leaving it to TRANSMIT to convert it into node and user ID by the nicknames section of the NAMES data set.

```
transmit joe da('sys1.parmlib') mem(ieasys00) line
ENTER MESSAGE FOR NODEB.USER2
Joe,
   These are the parmlib members you asked me to send you.
They are in fact the ones we are running today.
Yours, John Doe
<null line>
```

The message text in this example was entered in line mode which would be unusual for a user on a 3270 terminal, but which is easier to show in an example.

Example 4

Joe begins the receive process for the members transmitted in "Example 3" on page 346 and ends the receive without actually restoring the data onto the receiving system, because Joe does not know where he wants to store the data.

```
receive
Dataset SYS1.PARMLIB  from USER1 on NODEA
Member: IEASYS00
Greetings from John Doe.
Joe,
   These are the parmlib members you asked me to send you.
They are in fact the ones we are running today.
Yours, John Doe
NODEA.USER1
Enter restore parameters or 'DELETE' or 'END' +
end
```

In the preceding example, notice that the PROLOG and EPILOG lines have been appended to the message entered by the sender. In an actual RECEIVE operation, the original message text would appear in both uppercase and lowercase just as the sender had entered it (assuming the receiver's terminal supports lowercase.)

Example 5

Joe receives the 'SYS1.PARMLIB' members transmitted in Example 3. Specify space parameters for the data set that will be built by RECEIVE to leave space for later additions.

```
receive
Dataset SYS1.PARMLIB from USER1 on NODEA
Member: IEASYS00
Greetings from John Doe.
Joe,
   These are the parmlib members you asked me to send you.
They are in fact the ones we are running today.
Yours, John Doe
NODEA.USER1
Enter restore parameters or 'DELETE' or 'END' +
da('nodea.parmlib') space(1) cyl dir(10)
Restore successful to dataset 'NODEA.PARMLIB'
------------------------------------------------
No more files remain for the RECEIVE command to process.
```

The received member IEASYS00 is saved in the output data set with their member names unchanged.

Example 6

Send a message to a user on another system.

```
transmit davy
```

The system displays the following screen for input:

```
                    DATA FOR ALAMO.CROCKETT
0001   Davy,
0002     Did you check the report I gave you last week?
0003   Joe
0004
0005
:
```

Press PF3 to send the message.

In this example, the target user is identified by his nickname and no data set is specified, causing the terminal to be used as an input source. You can type your data, scroll using program function (PF) keys PF7 or PF19 and PF8 or PF20, and exit using PF3 or PF15, or cancel using the PA1 key.

Example 7

Send a member of a partitioned data set as a message and log the transmission in the data set CNFDNTL.MYLOG. In this example, the member MEETINGS of the partitioned data set MEMO.TEXT is sent as a message to JOE and this message is logged in 'MIKE.CNFDNTL.MYLOG'.

```
transmit nodeb.joe msgds(memo.text(meetings))  logda(cnfdntl.mylog)
 INMX000I 0 message and 7 data records sent as 5 records to NODEB.JOE
 INMX001I Transmission occurred on 07/27/87 at 09:00:35.
READY
```

JOE receives the message in his data set MY.LOG, instead of the default log data set, LOG.MISC:

```
receive logds(my.log)
 INMR901I Dataset ** MESSAGE ** from MIKE on NODED
 THIS IS A SCHEDULE OF STATUS MEETINGS FROM AUGUST THROUGH NOVEMBER:

 AUGUST      MONDAYS AT 9:00 A.M. IN MY OFFICE
 SEPTEMBER   TUESDAYS AT 10:00 A.M. IN YOUR OFFICE
 OCTOBER     WEDNESDAYS AT 10:00 A.M. IN JACK'S OFFICE
 NOVEMBER    MONDAYS AT 2:00 P.M. IN JILL'S OFFICE
```

TSOEXEC command

Use the TSOEXEC command to invoke an authorized command from an unauthorized environment. For example, you can use TSOEXEC when in the Interactive System Productivity Facility (ISPF), which is an unauthorized environment, to invoke authorized commands such as TRANSMIT and RECEIVE.

Three CLIST control variables are related to the use of the TSOEXEC command:

- &SYSABNCD contains the ABEND code.
- &SYSABNRC contains the ABEND reason code.
- &SYSCMDRC contains the command return code returned by the command most recently invoked by TSOEXEC.

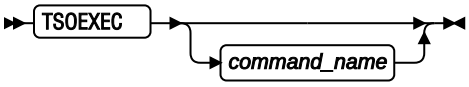
For more information about these variables, see *z/OS TSO/E CLISTs*.

These variables are changed slightly when used in REXX execs. They are as follows:

- SYSABNCD
- SYSABNRC
- SYSCMDRC

Note: Using TSOEXEC ISPSTART does not give a controlled environment. For information about controlled environments, see *z/OS Security Server RACF Security Administrator's Guide*.

TSOEXEC command syntax



TSOEXEC command operand

[*command_name*]
specifies any TSO/E command the TSO/E service facility can invoke, whether the command is authorized or unauthorized.

TSOEXEC command return codes

Table 57 on page 348 lists all the return codes of TSOEXEC command.

Table 57: TSOEXEC command return codes

Return code	Meaning
0	Processing successful.
4	Processing completed, but the requested command returned a non-zero return code. It is in CLIST control variable &SYSCMDRC.
8	An attention interruption ended the requested command.
12	The requested command abnormally terminated. Its abend code and REASON code are in CLIST control variables &SYSABNCD and &SYSABNRC.
24	System error.
28	The requested command is not a valid TSO/E command.

TSOEXEC command examples

Example 1

Operation: Use the TRANSMIT command to send a copy of a data set to another user while operating in ISPF.

Known:

- The user node: NODEB
- The user ID: USER2
- The data set name: SYS1.PARMLIB

```
TSOEXEC  TRANSMIT NODEB.USER2 DA('SYS1.PARMLIB')
```

TSOLIB command

The TSOLIB command provides for an additional search level that TSO/E uses when searching for commands and programs. With TSOLIB, you specify load module and program object libraries containing executable commands and programs, which are put to the top of the standard search order.

You can activate and deactivate the additional search level without leaving your TSO/E session. For the life of the additional search level, the activated load module and program object libraries serve as a task library to commands and programs you invoke.

This provides for flexible access to different versions of commands and programs, reduces the access time, and can simplify management of user IDs and LOGON procedures.

- The TSOLIB command, with its ACTIVATE operand, allows you to request access to load module and program object libraries. The requested load module and program object libraries will be put to the top of the system's search order for load module and program object libraries.
- It allows you, with its DEACTIVATE operand, to remove these load module and program object libraries from the search chain, thus, reestablishing the previous search order.
- The TSOLIB command allows for stacking multiple requests for load module and program object libraries, making any further request to become the active one but keeping the previous requests stacked for later use. Every removal, and respective deactivation, of the currently active request reactivates the previous request. This allows for faster variation of your library search order without having to enter lengthy command strings.

  The stacking of multiple requests can be inhibited, thus ensuring that a request is performed only if no previous request is active.
- The DISPLAY operand of the TSOLIB command shows the currently active libraries being put to the top of the standard search order. If any library requests are stacked, they are shown as well.
- The RESET operand sets the search order back to its original state.

Search order for load modules

The TSOLIB command is meant to provide a flexible way to extend the system's search order for commands and programs you invoke, or commands and programs invoked from other commands and programs.

For efficient use of the TSOLIB command you need to be aware of the search order, its variations through TSOLIB, and further variations by programs like ISPF that establish their own task libraries.

The standard search order

Without having used the TSOLIB command, TSO/E searches for a command or program using the following sequence:

1. The step library or job library

   The user's LOGON procedure is checked for any //STEPLIB DD-card that specifies a user's load module library or list of libraries. If the module is found here, it will be executed.
2. The link pack area

   The search is continued in the libraries specified in SYS1.PARMLIB member LPALSTnn. If the module is found here, it will be executed.
3. The link list concatenation

   The search is continued in the libraries specified in SYS1.PARMLIB member LNKLSTnn. The module should be found here.

Extending the range of a search with TSOLIB

With the first invocation of TSOLIB, you activate an additional search level and specify a load module library or a list of load module libraries. The specified libraries serve as a task library for further command and program invocations.

The system starts searching an invoked command or program in the task library you have activated. If a command or program is found in the newly activated libraries, it is executed; else the system follows the standard search order as described before.

The extended search order remains intact until one of the following happens:

- You reset the additional search level.

  The system will use the standard search order.
- You deactivate the additional search level.

  If you did not stack any previous requests, the system will use the standard search order.

  If a request has been stacked, the previously stacked request becomes active.
- You logoff from the session.

  After a new logon, the system will use the standard search order.
- You invoke an application, like ISPF, that places its own task libraries on top of the search order TSOLIB has set up.

  When that application completes, the search order TSOLIB has set up again becomes the top of the search chain.

Further considerations

- Authorized Commands and Programs

  A load module library activated by the TSOLIB command can contain unauthorized and authorized commands and programs.

  Authorized commands and programs:

  – Must have been link-edited with an authorization code of 1
  – Must reside in an APF-authorized library
  – Must be listed on the AUTHCMD, AUTHPGM or AUTHTSF statements of SYS1.PARMLIB member IKJTSOxx before they can be invoked.

  To allow authorized commands or programs to be invoked, the entire concatenation of data sets must contain data set names that reside in the APF-authorized library list. This means that, if you activate a list of load module and program object libraries with TSOLIB, every data set name representing a library must be named in the APF-authorized library list.
- Direct entry from applications to TSO/E

  Several applications, like ISPF, allocate their own task libraries, for example ISPLLIB, to be on top of the search order that TSOLIB set up. However, applications also have a direct entry in to TSO/E environment. For example:

  – TSO/E service facility in an isolated environment
  – The TSO/E TSOEXEC command
  – Authorized commands and programs

  These use the search order that TSOLIB has set up.
- Access permission to libraries

  If you have a security server active on your system, ensure that you are permitted read or execute access to the libraries TSOLIB is to activate.

Command usage

The TSOLIB command with its ACTIVATE, DEACTIVATE, and RESET operands is intended to be issued from TSO/E READY mode, either in the foreground or in the background. The requested extension on the search order becomes effective when TSO/E READY mode processes its next command. If the TSOLIB command is issued from any other environment, like ISPF or REXX, only the TSOLIB command with its DISPLAY operand is valid.

The TSOLIB command must be issued from a "TSO/E READY" environment. This condition is met when TSOLIB is invoked from the READY prompt or from a CLIST invoked from the READY prompt. To invoke the TSOLIB command from a REXX exec, however, the command must be placed on the REXX external data

stack. Execution is delayed until after the REXX exec completes. Just before the READY prompt is redisplayed, all of the commands on the REXX external data stack will be read and executed. This will retrieve and execute the TSOLIB command that was placed on the REXX external data stack by the REXX exec. For more information, see "TSOLIB command examples" on page 354.

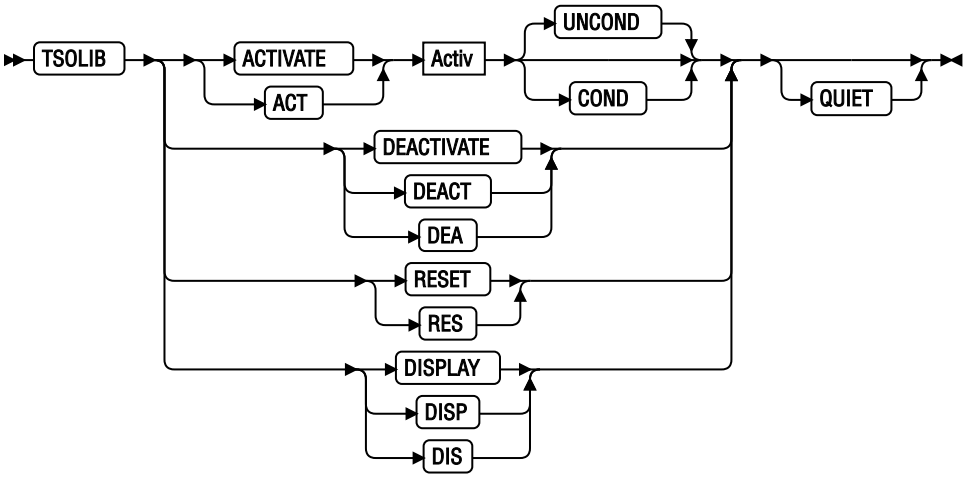
Stacking load module and program object library requests

Requests to activate load module and program object libraries into the search chain can be stacked. You control this with the COND and UNCOND operands of the TSOLIB command.

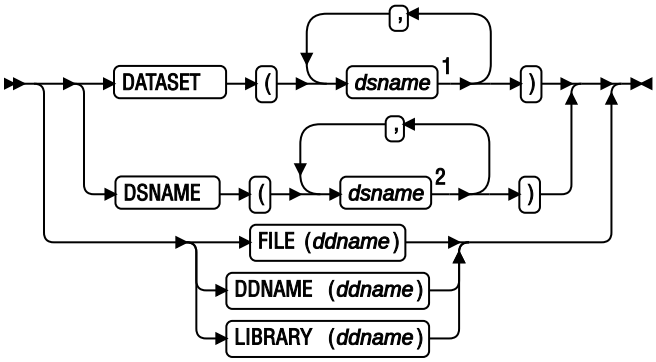
By default, a request to activate libraries is done unconditionally. The request becomes the current and active one. Any previous request (if one was issued) is stacked and temporarily made inactive. The next request to *deactivate* libraries will remove the current and active one from the search chain and re-activate the previous request (if there was one).

If a request is made conditionally, by using the COND keyword operand, the request will not become active if any previous activation took place before. See "TSOLIB command examples" on page 354 for a detailed example on how to stack load module library requests.

TSOLIB command syntax



Activ



Notes:

[1] 1 to 15 data sets for libraries.
[2] 1 to 15 data sets for libraries.

TSOLIB command operands

ACTIVATE | ACT
indicates that you want to include the specified libraries ahead of the standard search order.

DEACTIVATE | DEACT | DEA
indicates that you want to exclude the previously activated libraries from the top of the search order.

If previous activation requests have been done unconditionally, DEACTIVATE causes the last stacked request to become active again. See "Stacking load module and program object library requests" on page 351 for more information about stacking library activation requests.

DATASET(*dsname*[,*dsname*, ...]) | DSNAME(*dsname*[,*dsname*, ...])
specifies the data set name of a load module library, or a list of data set names of load module libraries, to be activated. Up to 15 data set names can be specified.

The data sets must be cataloged partitioned data sets, and they must be of the same record format (RECFM = U).

For the load module or program object libraries to be activated, the system automatically creates a ddname of SYS*nnnnn*. The ddname remains allocated until you issue TSOLIB DEACTIVATE or TSOLIB RESET.

If you want to activate more than 15 data set names, use the FILE operand of TSOLIB.

For authorized commands and programs to be invoked from a library read "Further considerations" on page 350.

FILE(*ddname*) | DDNAME(*ddname*) | LIBRARY(*ddname*)
specifies a *ddname* that represents a load module library or a list of load module libraries. The *ddname* must be allocated before you issue the TSOLIB command. The *ddname* remains allocated even after a TSOLIB DEACTIVATE or RESET command is issued. Use the FREE command to deallocate the ddname when required.

For authorized commands and programs to be invoked from a library read "Further considerations" on page 350.

Using a *ddname*, compared to a *dsname* or a list of *dsname*s, allows for a greater number of libraries to be activated. Use the ALLOCATE command to associate up to 255 data sets with a *ddname*; then issue `TSOLIB ACTIVATE FILE(`*ddname*`)`.

UNCOND | COND
controls the way TSOLIB is to treat an ACTIVATE request if previous requests have been performed.

UNCOND
(the default) indicates that the activation request is to be done unconditionally. Any active request is temporarily deactivated and stacked for later re-activation. See "Stacking load module and program object library requests" on page 351 for more information about stacking library activation requests, and "TSOLIB command examples" on page 354.

Note that stacked ddnames remain allocated. See also the description about the DATASET and FILE operands.

COND
indicates that the activate request is to be successful if no other request is active. Otherwise, the activate request is unsuccessful, a message is displayed, and a non-zero return code is set.

RESET | RES
excludes *all* specified libraries, set with the ACTIVATE operand, back to the standard search order. The search order for library load modules is now the same as it was before any TSOLIB command was given.

DISPLAY | DISP | DIS
issues information about the currently active *ddname* that is in front of the standard search order and those still on the stack, which will become the active ones, one after the other, with each following TSOLIB DEACTIVATE command.

If other task libraries became active after TSOLIB activated a library, for example, ISPF was started with ISPLLIB, the DISPLAY operand issues information about the situation.

QUIET
indicates that you do not want messages from this invocation of the TSOLIB command displayed.

The QUIET operand is primarily intended for programs under ISPF that invoke the TSOLIB command. The programs need access to the messages that TSOLIB issues, but will not want to display them. Trapping of messages is not available, and &SYSOUTTRAP cannot be used in a program.

If ISPF is active, the messages are saved in ISPF shared pool variables:

- Variable IKJTSM contains the number of non-blank messages being returned from this invocation of the TSOLIB command with the QUIET operand.
- Variable IKJTSM1 contains the first message, IKJTSM2 the second message, and so on. Up to 99 messages are saved in variables IKJTSM1 through IKJTSM99.

  The variables contain the actual messages that TSOLIB would have displayed if invoked without the QUIET operand. The lengths of the messages are not restricted to 80 characters.

The ISPF shared pool variables are only set when needed. They are not blanked out when not needed.

QUIET does not take effect until after the content of the command buffer, holding this invocation of the TSOLIB command, is known to be syntactically correct. If the command parser finds an error, or needs to prompt for input, it will issue messages and obtain input from the terminal as necessary.

Note: Do not use the QUIET option of TSOLIB in the IPCS dialog. IPCS does not make ISPF services available to TSO/E commands that IPCS invoke.

If you invoke the TSOLIB command without specifying an operand, TSOLIB will assume the ACTIVATE and DATASET operands and prompt you for the missing information. Note that prompting restricts you to a single data set name. You cannot enter a list of *dsname*s.

Entering only the significant characters can abbreviate operands. However, you might need to clarify reasons for the abbreviations shown.

TSOLIB command return codes

Table 58 on page 353 lists all the return codes of TSOLIB command.

Table 58: TSOLIB command return codes

Return code	Meaning
0	Processing successful. A load module or program object library, or a list of load module libraries, has been successfully activated, deactivated, or reset. However, informational messages may have been issued, for example, `IDY00020I Unable to free previously allocated data sets. Enter ? for more information.`
4	A TSOLIB library does not exist for this type (when deactivating a TSOLIB library).
8	A load module library already exists for this type when the COND operand is used.
16	The load module library specified with the TSOLIB ACTIVATE command was not previously allocated.
20	Severe error. More information is contained in the messages.
24	Internal processing error. TSOLIB is either unable to establish a recovery environment or encountered an error processing a TSOLIB installation exit.
28	Environment error. TSOLIB was not invoked in a TSO/E READY environment.
32	Environment error. TSOLIB was not invoked as a command processor.

TSOLIB command examples

Example 1: Activate a single data set

Operation: Activate a single data set `'sys3.loadlib1'` using the ACTIVATE DATASET operand of TSOLIB. Use the TSOLIB DISPLAY operand to display the current search order. Assume no previous request has been issued before.

```
TSOLIB ACTIVATE DATASET('sys3.loadlib1')
TSOLIB DISPLAY
IDY00022I Search order (by DDNAME) is:
IDY00023I DDNAME = SYS00101
```

Note that the system has created a ddname of SYS00101 for the activated load module library.

Example 2: Activate a concatenation of data sets

Operation: Activate data sets `'sys3.loadlib1'` and `'sys3.testlib'` using the ACTIVATE DATASET operand of TSOLIB. Use the TSOLIB DISPLAY operand to display the current search order. Assume no previous request has been issued before.

```
TSOLIB ACTIVATE DATASET('sys3.loadlib1' 'sys3.testlib')
TSOLIB DISPLAY
IDY00022I Search order (by DDNAME) is:
IDY00023I DDNAME = SYS00101
```

Note that the system has created a ddname of SYS00101 for the concatenated load module libraries.

Example 3: Activate an allocated File

Operation: (1) Allocate data set *my.load* and specify the ddname `aalib` to be associated with it, (2) activate ddname `aalib` with the TSOLIB ACTIVATE command, and (3) use the TSOLIB DISPLAY command to display to current search order. Assume no previous request has been issued before.

```
ALLOCATE FILE(aalib) DATASET(my.load)
TSOLIB ACTIVATE FILE(aalib)
TSOLIB DISPLAY
IDY00022I Search order (by DDNAME) is:
IDY00023I DDNAME = AALIB
```

Note: ALLOCATE command assumes OLD, which causes exlussive access, because it does not have NEW, OLD, MOD or SHR.

Example 4: Activate a data set from within a CLIST

Operation: Activate data set `'JIM.LOAD'` from within a CLIST running in TSO/E READY environment.

```
PROC 0
TSOLIB ACTIVATE DATASET('JIM.LOAD')
IF &LASTCC = 0 THEN +
   … process commands and programs from TSOLIB data set.
⋮
```

Example 5: Activate an allocated file from within a REXX Exec

Operation: (1) Allocate data set `'JIM.LOAD'` and specify the ddname `MYLOAD` to be associated with it, (2) activate ddname `aalib` with the TSOLIB ACTIVATE FILE command. Note that the REXX exec is to run in a TSO/E READY environment.

```
/* rexx */
"ALLOCATE FILE(MYLOAD) DATASET('JIM.LOAD') SHR"
if RC = 0 then
    push "TSOLIB ACTIVATE FILE(MYLOAD)"
exit
⋮
   … back in TSO/E READY environment, start the REXX exec
⋮
   … invoke commands and programs from TSOLIB data set.
⋮
```

Example 6: The use of TSOLIB library stacking

Operation: Activate data set `'sys3.loadlib1'` using the ACTIVATE DATASET operand of TSOLIB. Use the TSOLIB DISPLAY operand to display the current search order. Assume no previous request has been issued before.

```
TSOLIB ACTIVATE DATASET('sys3.loadlib1')
TSOLIB DISPLAY
IDY00022I Search order (by DDNAME) is:
IDY00023I DDNAME = SYS00101
```

Note that the system has created a ddname of SYS00101 for the activated load module library.

Operation: Activate another data set `'sys3.loadlib2'` using the ACTIVATE DATASET operand of TSOLIB. Use the TSOLIB DISPLAY operand to display the currently active and stacked ddnames.

```
TSOLIB ACTIVATE DATASET('sys3.loadlib2')
TSOLIB DISPLAY
IDY00022I Search order (by DDNAME) is:
IDY00023I DDNAME = SYS00102
IDY00024I DDNAME = SYS00101 (Stacked)
```

Note that the system has created a ddname of SYS00102 for the activated load module library. The previously activated ddname SYS00101 is temporarily deactivated and marked (Stacked).

Operation: (1) Allocate a third data set *my.load* and specify the ddname `aalib` to be associated with it, (2) activate ddname `aalib` with the TSOLIB ACTIVATE command, and (3) use the TSOLIB DISPLAY command to display the currently active and stacked ddnames.

```
ALLOCATE FILE(aalib) DATASET(my.load)
TSOLIB ACTIVATE FILE(aalib)
TSOLIB DISPLAY
IDY00022I Search order (by DDNAME) is:
IDY00023I DDNAME = AALIB
IDY00024I DDNAME = SYS00102 (Stacked)
IDY00024I DDNAME = SYS00101 (Stacked)
```

The previously activated ddname SYS00102 is temporarily deactivated and marked (Stacked) in addition to SYS00101. Ddname AALIB is the active library included ahead of the standard search order.

Example 7: The use of the TSOLIB COND operand

Operation: Based on “Example 6: The use of TSOLIB library stacking” on page 355, try to activate ddname `trylib` with the COND operand.

```
ALLOCATE FILE(trylib) DATASET(your.load)
TSOLIB ACTIVATE FILE(trylib) COND
IDY00015I TSOLIB terminated.  Load library already active
          and COND keyword was specified.
⋮
TSOLIB DISPLAY
IDY00022I Search order (by DDNAME) is:
IDY00023I DDNAME = AALIB
IDY00024I DDNAME = SYS00102 (Stacked)
IDY00024I DDNAME = SYS00101 (Stacked)
```

The activate request is unsuccessful; the previous activation remains unchanged.

Example 8: Reactivate a TSOLIB library from the stack

Operation: Based on “Example 6: The use of TSOLIB library stacking” on page 355, (or “Example 7: The use of the TSOLIB COND operand” on page 355) exclude the currently active library AALIB and activate the last one stacked (SYS00102).

```
TSOLIB DEACTIVATE
TSOLIB DISPLAY
IDY00022I Search order (by DDNAME) is:
IDY00023I DDNAME = SYS00102
IDY00024I DDNAME = SYS00101 (Stacked)
```

Example 9: The use of the TSOLIB QUIET operand

Operation: (1) Activate a single data set `'aalib.load'`, to which the system associates a ddname of SYS00100, (2) activate a concatenation of data sets, to which the system associates a ddname of SYS00101, (3) and invoke a REXX exec to show use of the QUIET operand when ISPF is active.

The contents of the variables in the ISPF shared pool are then examined. Assume no previous request has been issued before.

```
READY
TSOLIB ACTIVATE DATASET(mylib.load)
TSOLIB ACTIVATE DATASET('sys3.loadlib1' 'sys3.testlib')
```

Invoke the following REXX exec when ISPF is active:

```
/* rexx */
ADDRESS TSO
"TSOLIB DISPLAY QUIET"
⋮
```

The ISPF shared pool variables are now set as follows:

Variable
Content

IKJTSM
4

IKJTSM1
TSOLIB DISPLAY QUIET

IKJTSM2
IDY00022I Search order (by DDNAME) is:

IKJTSM3
IDY00023I DDNAME = SYS00101

IKJTSM4
IDY00024I DDNAME = SYS00100 (Stacked)

VLFNOTE command

When you change data that is shared across systems and managed by the virtual lookaside facility (VLF), you might need to enter the VLFNOTE command to notify VLF of the change. VLF needs to know when you make changes to the data it manages so that it can make current data available for users. The types of data VLF manages are:

- Data in a partitioned data set (PDS) or partitioned data set extended (PDSE)
- A named collection of data (non-PDS)

Note: The term partitioned data set (PDS) in the following VLFNOTE description refers to PDS and PDSE data sets.

The type of data and the system environment determine whether you need to enter VLFNOTE.

You do not need to use VLFNOTE (because notification to VLF is automatic) when both of the following are true:

- VLF is running on z/OS systems and are part of a single sysplex.
- The changed data belongs to a partitioned data set class.

When both conditions are true, VLF receives notification automatically through sysplex services. (*z/OS MVS Setting Up a Sysplex* describes running VLF in a sysplex.) Otherwise, you need to enter VLFNOTE.

The types of changes that require VLF notification are listed below:

For data in a PDS, enter VLFNOTE when you are:

- Adding a member to an eligible data set (a data set that is identified to VLF).
- Adding a member to a non-eligible data set when both of the following are true:
  - The new member is in a user's SYSPROC concatenation ahead of an eligible data set.
  - The eligible data set has a member with the same name as the new member.
- Updating an existing member of an eligible data set.
- Deleting an eligible data set or member of an eligible data set.

To notify VLF about changes to data in a PDS, use the VLFNOTE command syntax described in "Changing data associated with a partitioned data set" on page 357.

For non-PDS data, use VLFNOTE when you are:

- Adding a minor name to a major name.
- Updating a minor name associated with a major name.
- Deleting a minor name from a major name.

To notify VLF about changes to non-partitioned data, use the VLFNOTE command syntax described in "Changing non-PDS data" on page 358.

There are several ways to issue the VLFNOTE command. Depending on the method available at your installation you can:

- Logon to each of the other systems in your complex and enter VLFNOTE.
- Send a message to a user on each of the other systems in your complex and have them enter VLFNOTE.
- Submit a short batch job, with system affinity, to each of the other systems in your complex and issue VLFNOTE in the job.
- If your installation is using APPC/MVS, write an APPC/MVS transaction program to prompt the affected systems to issue the VLFNOTE command. Each of the affected systems must have an APPC/MVS transaction program that will issue the VLFNOTE command.

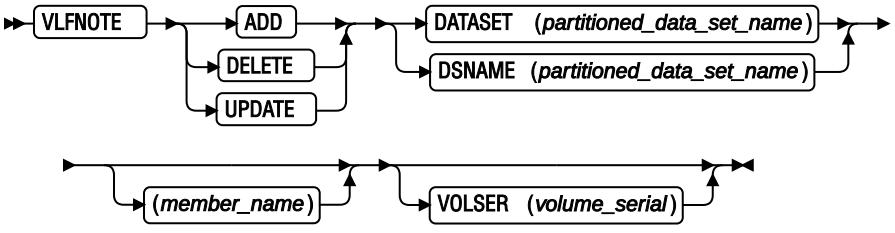
See *z/OS MVS Programming: Authorized Assembler Services Guide*, for more information about VLF notification.

Changing data associated with a partitioned data set

Use the following syntax to notify VLF that you have changed a partitioned data set. To notify VLF that you have changed non-PDS data, see "Changing non-PDS data" on page 358 for the correct syntax.

Note: For partitioned data set changes, the VLFNOTE command needs to be issued on each system in the shared DASD complex, except for the system on which the change was made. VLF is automatically notified on the system on which the change was made.

VLFNOTE command syntax (partitioned data set)



VLFNOTE command operands (partitioned data set)

ADD
specifies that you have added a member to a partitioned data set.

DELETE
specifies that you have deleted a partitioned data set or a member of a partitioned data set.

UPDATE
specifies that you have updated a member of a partitioned data set.

DATASET | DSNAME (*partitioned_data_set_name* [(*member_name*)])
specifies the name of the partitioned data set that you changed. Include the member name if you have added, deleted or updated a member.

VOLSER(*volume_serial*)
specifies the volume on which the changed partitioned data set resides. If you do not include VOLSER, VLF uses the catalog to determine the volume serial where the data set resides.

VLFNOTE command examples (partitioned data set)

Example 1

Operation: Notify VLF that you deleted member MAKEMEMO of partitioned data set 'COMMON.TOOLS.CLIST'

```
vlfnote delete dataset('common.tools.clist(makememo)')
```

Example 2

Operation: Notify VLF that you renamed a member, X, of a CLIST data set "YOURID.TAILORED.CLIST". It is now called NEWX. Delete the old member name and add the new member name.

```
vlfnote delete dataset(tailored.clist(X))
```

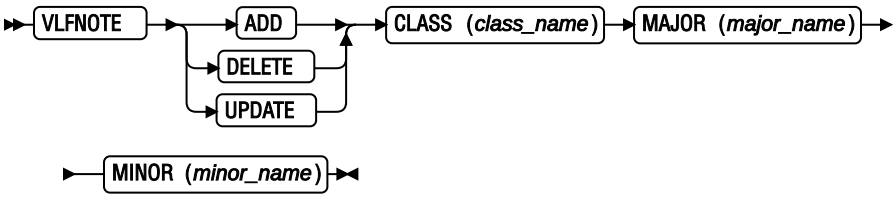
and

```
vlfnote add dataset(tailored.clist(newx))
```

Changing non-PDS data

Use this syntax to notify VLF that you changed non-PDS data (data that belongs to a CLASS-MAJOR or CLASS-MAJOR-MINOR combination). The specified class name must be an installation-supplied class name. To notify VLF that you changed a partitioned data set see "Changing data associated with a partitioned data set" on page 357 for the correct syntax.

VLFNOTE command syntax (non-PDS)



VLFNOTE command operands (non-PDS)

ADD
specifies that you have added a minor name to a major name.

DELETE
specifies that you have deleted a minor name from a major name or that you have deleted a major name from an installation-supplied class.

UPDATE
specifies that you have updated a minor name associated with a major name.

CLASS(*class_name*)
specifies the name of an installation-supplied class (class name beginning with a letter from H - Z) affected by the change you made.

MAJOR(*major_name*)
specifies the major name associated with the change you made.

MINOR(*minor_name*)
specifies the minor name associated with the change you made.

VLFNOTE command examples (non-PDS)

Example 1

Operation: Notify VLF that non-PDS data has been deleted.

```
vlfnote delete class(myclass1) major(major1) minor(minor1)
```

Example 2

Operation: Notify VLF that you deleted major name "NOTICE", of the installation-supplied class "MYCLASS".

```
vlfnote delete class(myclass) major(notice)
```

VLFNOTE command return codes

Table 59 on page 359 lists the return codes of VLFNOTE command.

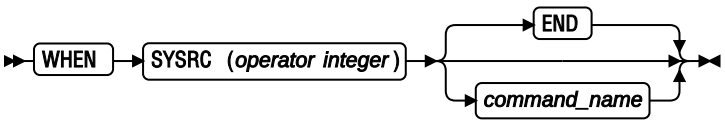
Table 59: VLFNOTE command return codes

Return code	Meaning
0	Processing successful.
12	Return code 12 means one of the following: • Incorrect syntax was specified for the command. • The invoked VLF function returned a non-zero return code. • The TSO/E parse service routine or the TSO/E catalog information routine returned a non-zero return code. • Unauthorized for specific request. Error messages indicate the exact problem.

WHEN command

Use the WHEN command to test return codes from programs invoked by an immediately preceding CALL or LOADGO command, and to take a prescribed action if the return code meets a certain specified condition.

WHEN command syntax



WHEN command operands

SYSRC
specifies the return code from the previous function (the previous command in the CLIST) 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 processing is to be terminated if the comparison is true. If you do not specify a command, END is the default.

command_name
specifies any valid TSO/E command name and appropriate operands. If the comparison is true, TSO/E processes the command.

WHEN terminates CLIST processing and then executes the TSO/E command name specified.

Use successive WHEN commands to determine an exact return code and then perform some action based on that return code.

WHEN command return code

The return code is from the command that executed last.

WHEN command examples

Example 1

Operation: Use 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
```

Chapter 2. Session Manager commands

This chapter describes the functions and syntax of each Session Manager command. It includes:

- The general format and syntax rules for the commands.
- A description of the function and syntax for each command. The commands are described in alphabetical order.

Introductory information about how to use and start Session Manager at your terminal is described in *z/OS TSO/E User's Guide*.

Entering Session Manager commands

You can enter Session Manager commands by:

- Pressing the CLEAR key and entering a command anywhere on the screen
- Pressing a program function (PF) key set up to issue a command
- Executing a TSO/E CLIST, which contains commands
- Defining the command as the text_string of the SMPUT command

Regardless of how you enter the commands, the following rules apply:

- You can enter multiple Session Manager commands on one line provided you separate them with a semicolon (;). The number of characters on any one line cannot exceed 512. When multiple commands are entered on a line, an error in one command does not prevent the remaining commands from executing.
- In order for a Session Manager command to execute, it must be placed in the SMIN stream.
- Any change you make to the definitions of the windows, cursor, PF keys, session functions, streams, or the terminal, remain in effect for your terminal session. You can place these definitions in a CLIST to be executed each time you log on.

Command format

A Session Manager command consists of a *command name*, typically followed by a *command modifier* and one or more *operands*.

A command name is typically a familiar English word that describes the function of the command. Some command names are followed by command modifiers, which qualify the action of the command name.

Operands provide specific information about how an operation is to be performed. The two types of operands used with Session Manager commands are *positional* and *keyword*.

`command.modifier`	one or more blanks	`operand, operand, ...`

You must separate the command name and command modifier with a period or one or more blanks. Also separate the command or command modifier from the first operand by one or more blanks.

The command descriptions include some operands in lowercase letters and some in uppercase letters. If the operand is in lowercase letters, you must substitute a specific value for the letters. For example, in the command

```
CHANGE.PFK pfk_number …
```

you must replace *pfk_number* with the number of the program function (PF) key to be changed.

```
CHANGE.PFK 1 …
```

Lowercase operands are *positional* operands because they follow the command names or modifiers in a prescribed order.

The operands in uppercase letters are *keyword* operands. You must type those operands as shown. For example:

```
ALARM(ON)

CONTROL(seconds)
```

Both of the preceding keyword operands have subfield values enclosed in parentheses. You must type the keyword (or its abbreviation), a left parenthesis, and then the subfield value. You can omit the closing parenthesis if it is the last character of the command.

The ALARM keyword shows the only possible subfield value within the parentheses.

The CONTROL operand shows the subfield value in lowercase letters. Therefore, you are to substitute a value for the lowercase name.

Session Manager Command syntax

The command syntax for Session Manager commands is represented using structured diagrams. This method of syntax representation is described in “How to read the TSO/E command syntax” on page 2.

Defaults

To make the commands easier to enter, certain operands default to a specific value. If the default value is the value you want to use, you do not have to enter the operand. The default values are underlined in the syntax description for each command.

Many Session Manager commands see a *default window*. The default window is the MAIN window or the window you have assigned as the default window via the CHANGE.TERMINAL command.

Abbreviations

You can abbreviate nearly all Session Manager command names, modifiers, and keyword operands. These abbreviations can be as short as possible, while still providing uniqueness among them. For example, the minimum abbreviation for the DELETE command is:

```
DEL
```

DEL distinguishes DELETE from the DEFINE command.

The exceptions to this rule are the following frequently used commands:

```
DEFINE    D
SCROLL    S
RESTORE   R
```

You can also abbreviate any keyword operand. For example, the keyword operands of the CHANGE.WINDOW command and their minimum abbreviations are:

```
ALARM     A
HOLD      H
OVERLAP   O
PROTECT   P
```

```
TARGET    T
UPDATE    U
VIEW      V
```

The Session Manager also accepts the following commonly used abbreviations:

```
CONTROL   CNTL
FORMAT    FMT
```

Session Manager Command summary

Table 60 on page 363 summarizes the Session Manager commands and their functions.

Table 60: Summary of the Session Manager commands

Command	Function
CHANGE.CURSOR	Changes the permanent or temporary location of the cursor.
CHANGE.FUNCTION	Changes whether the terminal's audible alarm is to sound when information enters an input or output stream.
	Specifies whether information from an input stream is to be copied to an output stream and the intensity at which the information is to be displayed.
	Specifies the input stream for a session function.
	Specifies the output stream for a session function and the intensity at which the information is to be displayed.
CHANGE.MODE	Indicates whether you want to run under VS/APL or under the Session Manager.
CHANGE.PFK	Changes the definition of a program function (PF) key.
CHANGE.STREAM	Specifies whether the terminal's audible alarm is to sound when information enters a given stream.
	Erases all of the information in a given stream.
CHANGE.TERMINAL	Specifies whether the terminal's audible alarm is to sound when the keyboard unlocks.
	Specifies the maximum time the keyboard is to be locked while a command is executing.
	Changes the default window.
CHANGE.WINDOW	Specifies whether the terminal's audible alarm is to sound when the Session Manager scrolls a window to display new information.
	Specifies how long a window is to be held in place before the Session Manager scrolls it.
	Specifies how many lines of a window's old position are to be repeated when the window scrolls to a new position.
	Specifies whether information can be entered in a window.
	Indicates the name of the stream that is to receive the information in a window and the intensity at which the information is to be displayed.
	Indicates how much new information must enter a stream before the window scrolls to display it.
	Specifies the name of the stream a window is to display.
DEFINE.WINDOW	Defines a new window on the display screen.
DELETE.WINDOW	Deletes a window from the display screen.

Table 60: Summary of the Session Manager commands (continued)

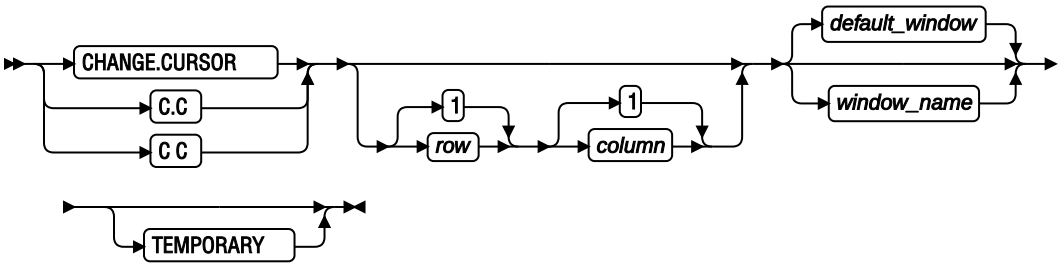
Command	Function
END	Ends Session Manager support of your TSO/E session.
FIND	Searches for a text string in a stream that is currently displayed by a window. Finds the number of the top line being displayed by a window.
PUT	Places a text string in a stream and indicates the intensity at which the text string is to be displayed.
QUERY	Displays information about: • TSO, SM, and MSG functions • Program function (PF) keys • Streams • Terminal • Windows
RESET	Restarts the Session Manager display environment.
RESTORE	Restores the definitions of the following, which were saved through the SAVE command: • Program function (PF) keys • Screen layout • Windows
SAVE	Saves the definitions of the following: • Program function (PF) keys • Screen layout • Windows
SCROLL	Moves a window over a stream.
SNAPSHOT	Copies a display screen of information into a stream.
UNLOCK	Unlocks a window.

CHANGE.CURSOR command

Use the CHANGE.CURSOR command to change the location of the cursor on the display screen. You can establish a *permanent* or *temporary* location for the cursor.

If you define a permanent location, the cursor returns to that location each time you press a program function (PF) key, the Enter key, the attention (PA1) key, the CLEAR key or the cancel (PA2) key. If you define a temporary location, the cursor moves to and remains at that location until the next keyboard entry. After the keyboard entry, the cursor moves to the permanent location.

CHANGE.CURSOR command syntax



CHANGE.CURSOR command operands

row column
The location in the specified window where the cursor is to go. The *row* and *column* numbers are relative to those in the window, not the entire display screen. If you specify a number for *row* that is greater than the number of lines in the window, the Session Manager uses the last row in the window. If you specify a number for *column* that is greater than the number of columns in the window, the Session Manager adjusts the column value to the number of columns in the window minus one. If you specify 0 or a negative number for *row* or *column*, the Session Manager places the cursor in the first row and column in the window.

window_name
The name of the window where the cursor is to be placed.

TEMPORARY
specifies that this change to the cursor location is to be temporary. If both the row and column and window_name operands are omitted, and a temporary location for the cursor was previously set, the Session Manager moves the cursor to that location. If a temporary location was not previously set, the Session Manager moves the cursor to the upper left hand corner of the display screen.

Unless you specify TEMPORARY, the change is permanent.

CHANGE.CURSOR command return codes

Table 61 on page 365 lists the return codes of CHANGE.CURSOR command.

Table 61: CHANGE.CURSOR command return codes

Return codes	Meaning
0	Processing successful.
4	Syntax error in command.
8	Window not found.

CHANGE.CURSOR command examples

Example 1

Set the permanent location of the cursor to row 5, column 3 of the TEST window.

```
change.cursor 5 3 test
```

Example 2

Set the temporary location of the cursor to row 2, column 1 of the ENTRY window.

```
change.cursor 2 1 entry temporary
```

Example 3

Set the cursor to the temporary location that was set in a previous CHANGE.CURSOR command.

```
change.cursor temporary
```

Example 4

Change the permanent location of the cursor to row 1, column 1 of the default window.

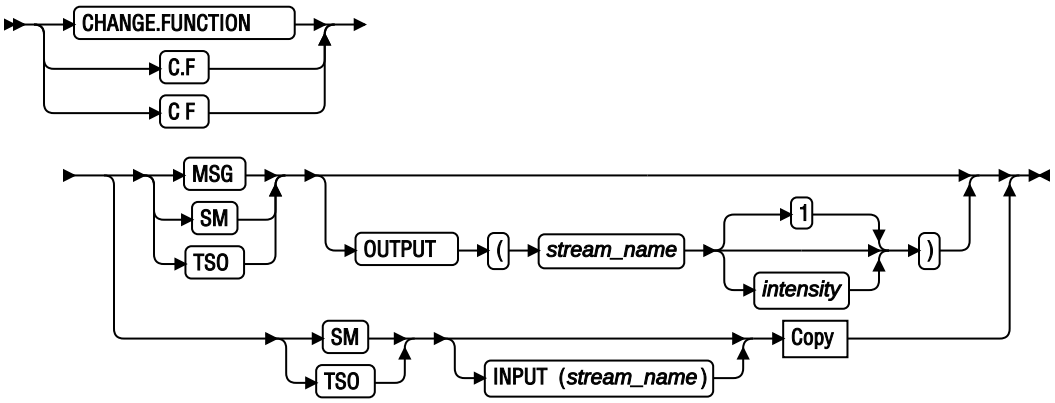
```
change.cursor
```

CHANGE.FUNCTION command

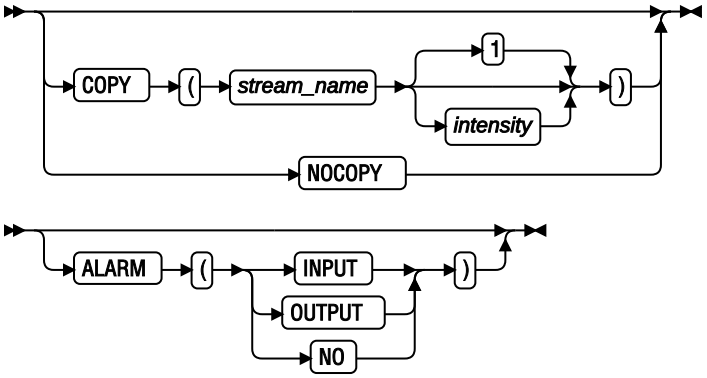
Use the CHANGE.FUNCTION command to:

- Change whether the terminal's audible alarm is to sound when information enters an input or output stream
- Specify whether information from an input stream is to be copied to an output stream and the intensity at which the information is to be displayed
- Specify the input stream for the TSO, Session Manager (SM), or message (MSG) functions
- Specify the output stream for a session function and the intensity at which the information is to be displayed

CHANGE.FUNCTION command syntax



Copy



CHANGE.FUNCTION command operands

MSG
requests that the change apply to the message (MSG) function. The MSG function represents the messages from other TSO/E users, the operator, and background jobs.

SM
requests that the change apply to the Session Manager (SM) function. The SM function represents work related to the Session Manager.

TSO
requests that the change apply to the TSO function. The TSO function represents work related to TSO.

OUTPUT(*stream_name intensity*)

stream_name
The name of the output stream for the specified function.

intensity
specifies the brightness at which the information is to be displayed in the output stream. The valid values are:

1
The information is to be displayed at normal intensity.

2
The information is to be highlighted.

INPUT(*stream_name*)
The name of the input stream for the specified function.

COPY(*stream_name intensity*) | NOCOPY

COPY(*stream_name intensity*)
requests that the Session Manager copy the input stream for this function into an output stream.

stream_name
is the name of the output stream that is to contain a copy of the information from the input stream.

intensity
specifies the brightness at which the copied information is be displayed. The valid values are:

0
The copied information is not to be displayed. You can see the line that the information occupies, but the information itself is invisible.

1
The copied information is to be displayed at normal intensity.

2
The copied information is to be highlighted.

NOCOPY
specifies that the Session Manager is not to copy the information from the input stream into an output stream.

ALARM(INPUT | OUTPUT | NO)
specifies whether the terminal's audible alarm is to sound when information enters a stream. The stream does not have to be currently displayed for the alarm to sound. You can set ALARM to sound for either the input stream or the output stream, but not both.

Note: If your terminal does not have an audible alarm, the Session Manager still accepts this operand. It has no way of knowing whether your terminal has an audible alarm.

INPUT
specifies that the audible alarm is to sound when a line of information enters the input stream for the specified function.

OUTPUT
specifies that the audible alarm is to sound when the a line of information enters the output stream for the specified function.

NO
specifies that the audible alarm is not to sound when information is added to any of the function streams.

CHANGE.FUNCTION command return codes

Table 62 on page 368 lists the return codes of CHANGE.FUNCTION command.

Table 62: CHANGE.FUNCTION command return codes	
Return codes	**Meaning**
0	Processing successful.
4	Syntax error in command.
8	Stream not found.

CHANGE.FUNCTION command examples

Example 1

Define the MSG function so that all messages are highlighted in the TSOOUT stream and the terminal's audible alarm sounds when you receive a message.

```
change.function msg output(tsoout 2) alarm(output)
```

Example 2

Set the Session Manager function to get commands from the SMIN stream and place the output from these commands in the SMOUT stream highlighted. The information in the SMIN stream is not to be copied to the SMOUT stream.

```
change.function sm input(smin) output(smout 2) nocopy
```

Example 3

Set the SM function to copy the SMIN stream into the SMOUT stream.

```
change.function sm copy(smout)
```

Example 4

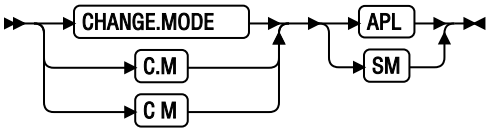
Set the TSO function to get its input from the TSOIN stream and highlight its output in the TSOOUT stream. The information in the TSOIN stream is to be copied into the TSOOUT stream.

```
change.function tso input(tsoin) output(tsoout 2)
  copy(tsoout)
```

CHANGE.MODE command

Use the CHANGE.MODE command to indicate whether you want to run under VS/APL or the Session Manager.

CHANGE.MODE command syntax



CHANGE.MODE command operands

APL
indicates that you are running VS/APL and you want the Session Manager to provide any additional functions that were designed specifically to enhance the interface between the Session Manager and VS/APL.

Note: To use a VS/APL program function (PF) key definition, first make the corresponding Session Manager PF key definition null. To find out how to make a PF key null, see the CHANGE.PFK command.

SM
indicates that you are running under the Session Manager.

CHANGE.MODE command return codes

Table 63 on page 369 lists the return codes of CHANGE.MODE command.

Table 63: CHANGE.MODE command return codes

Return codes	Meaning
0	Processing successful.
4	Syntax error in command.

CHANGE.MODE command examples

Example 1

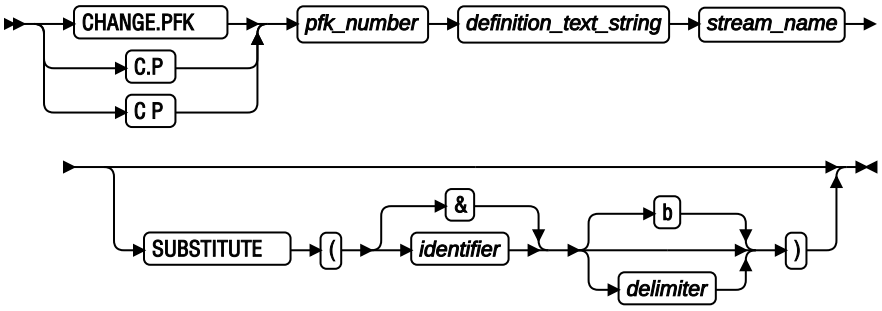
Change the mode to run under VS/APL.

```
change.mode apl
```

CHANGE.PFK command

Use the CHANGE.PFK key to change the definition of a program function (PF) key. You can define a PF key to issue one or more Session Manager commands, TSO/E commands, input to an application program, or any other string of characters.

CHANGE.PFK command syntax



CHANGE.PFK command operands

pfk_number
The number of the PF key to be changed. If you specify a number that does not exist on your terminal, the Session Manager still accepts this operand. It has no way of knowing how many PF keys you have.

definition_text_string
The string of characters that are to be placed in the specified stream. If the text string contains lowercase letters, blanks, commas, or parentheses, enclose it in single quotation marks. A single quotation mark in the text string must be represented as two adjacent quotation marks.

If there are no blanks, commas, or parentheses in the text string, you can omit the enclosing quotation marks. If you omit the quotation marks, however, the Session Manager translates the text string to uppercase letters. When using the CHANGE.PFK command in a CLIST, the Session Manager always stores the text string in uppercase letters, even if it is enclosed in quotation marks.

If you enter more than one command for the text string, separate them with a semicolon (;).

If you want to use a PF key defined under some other 3270 application (for example, VS/APL), first define the *definition_text_string* for the PF key as null to the Session Manager. The PF key can then be

passed back to the application. To specify a null PF key, define the *definition_text_string* as two adjacent quotation marks ('').

stream_name
The name of the stream where the text string is to be placed.

SUBSTITUTE
specifies that the information read from the screen is to be substituted into the '*definition_text_string*', replacing the symbolic arguments.

identifier
identifies the symbolic argument that is to be replaced. Any character (except a blank or comma) can be used as the identifier. If the identifier character appears elsewhere in the *definition_text_string*, it must be doubled.

delimiter
separates the information about the screen that is to be substituted into the text string. One or more blanks are treated as a single delimiter. The *delimiter* can be any character except a comma.

CHANGE.PFK command return codes

Table 64 on page 370 lists all the return codes of CHANGE.PFK command.

Table 64: CHANGE.PFK command return codes

Return codes	Meaning
0	Processing successful.
4	Syntax error in command.
8	Stream not found.

CHANGE.PFK command examples

Example 1

Change PF1 to place the TSO/E TIME command in the TSOIN stream where it will be executed.

```
change.pfk 1 'time' tsoin
```

Example 2

Change PF12 to issue the QUERY.TERMINAL command. Direct the output to the EXTRA1 stream and cause the default window to display that stream.

```
change.pfk 12 'query.terminal extra1;change.window
  view(extra1)' smin
```

Example 3

Change PF2 to issue the TSO/E LISTDS command. Each line typed just before the key is pressed is to be substituted as the data set name operand for the command.

```
change.pfk 2 'listds &1;.* members' tsoin substitute
```

If you type the following on the screen:

```
test
sample
```

and pressed PF2, the following TSO/E commands are executed:

```
listds test.* members
listds sample.* members
```

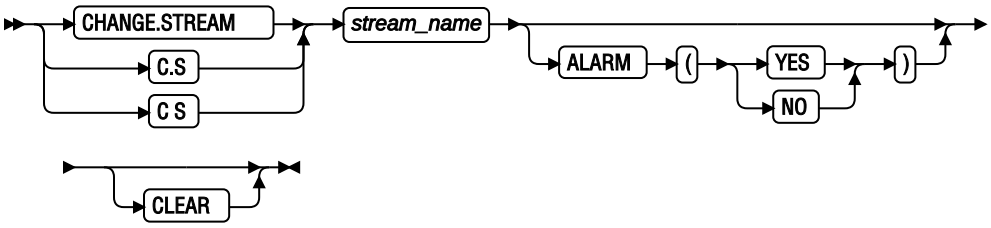
CHANGE.STREAM command

Use the CHANGE.STREAM command to:

- Change whether the terminal's audible alarm is to sound when information enters a stream
- Erase all of the information in a stream

(Use the QUERY.STREAMS command to display the names and attributes of all of the streams.)

CHANGE.STREAM command syntax



CHANGE.STREAM command operands

stream_name
The name of the stream to be changed.

ALARM(YES | NO)
specifies whether the terminal's audible alarm is to sound when information enters the stream. The stream does not have to be currently displayed for the alarm to sound. If your terminal does not have an audible alarm, the Session Manager still accepts this operand. It has no way of knowing whether your terminal has an alarm.

CLEAR
erases all of the information in the stream. The stream itself is not erased.

CHANGE.STREAM command return codes

Table 65 on page 371 lists all the return codes of CHANGE.STREAM command.

Table 65: CHANGE.STREAM command return codes

Return codes	Meaning
0	Processing successful.
4	Syntax error in command.
8	Stream not found.

CHANGE.STREAM command examples

Example 1

Set the display terminal's audible alarm to sound when information goes into the EXTRA1 stream.

```
change.stream extra1 alarm(yes)
```

Example 2

Erase all information in the TSOOUT stream.

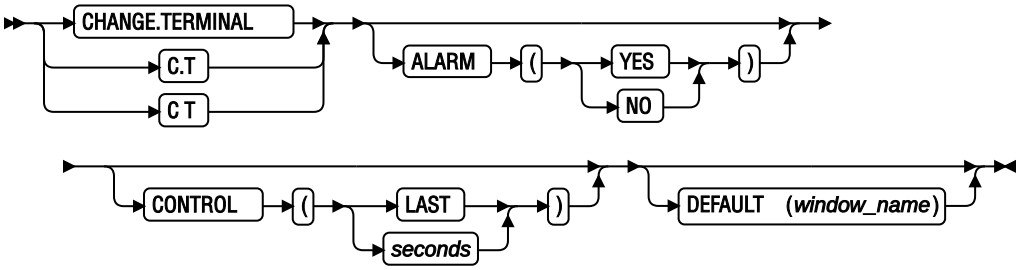
```
change.stream tsoout clear
```

CHANGE.TERMINAL command

Use the CHANGE.TERMINAL command to:

- Specify whether the audible alarm is to sound when the keyboard unlocks
- Indicate the maximum time the keyboard is to be locked while a command is executing
- Change the default window

CHANGE.TERMINAL command syntax



CHANGE.TERMINAL command operands

ALARM(YES | NO)
specifies whether the terminal's alarm is to sound when the keyboard unlocks. When the keyboard is unlocked, you can enter input. If your terminal does not have an audible alarm, the Session Manager still accepts this operand. It has no way of knowing whether your terminal has an alarm.

CONTROL(LAST | *seconds*)
specifies the maximum time, in seconds, that the keyboard is to remain locked. The Session Manager sets a timer to unlock the terminal keyboard when the time expires. Note, however, that when the keyboard is locked, you cannot enter commands, and attention interrupts will not be processed.

LAST
specifies that the timer is to be set to the last non-zero value entered for the CONTROL keyword of this command.

seconds
specifies that the keyboard is to be unlocked after the specified number of seconds has elapsed. *seconds* must be an integer from 0 to 999.

DEFAULT(*window_name*)
The name of the window that you want to be the default window. This window serves as the default window for other Session Manager commands when a window name is entered with the command. The MAIN window is the IBM-supplied default window.

CHANGE.TERMINAL command return codes

Table 66 on page 372 lists all the return codes of CHANGE.TERMINAL command.

Table 66: CHANGE.TERMINAL command return codes

Return codes	Meaning
0	Processing successful.

Table 66: CHANGE.TERMINAL command return codes (continued)	
Return codes	**Meaning**
4	Syntax error in command.
8	Window not found.

CHANGE.TERMINAL command examples

Example 1

Set the terminal so that the keyboard will be locked for no more than 10 seconds.

```
change.terminal control(10)
```

Example 2

Set the terminal so that each time the keyboard unlocks the audible alarm sounds.

```
change.terminal alarm(yes)
```

Example 3

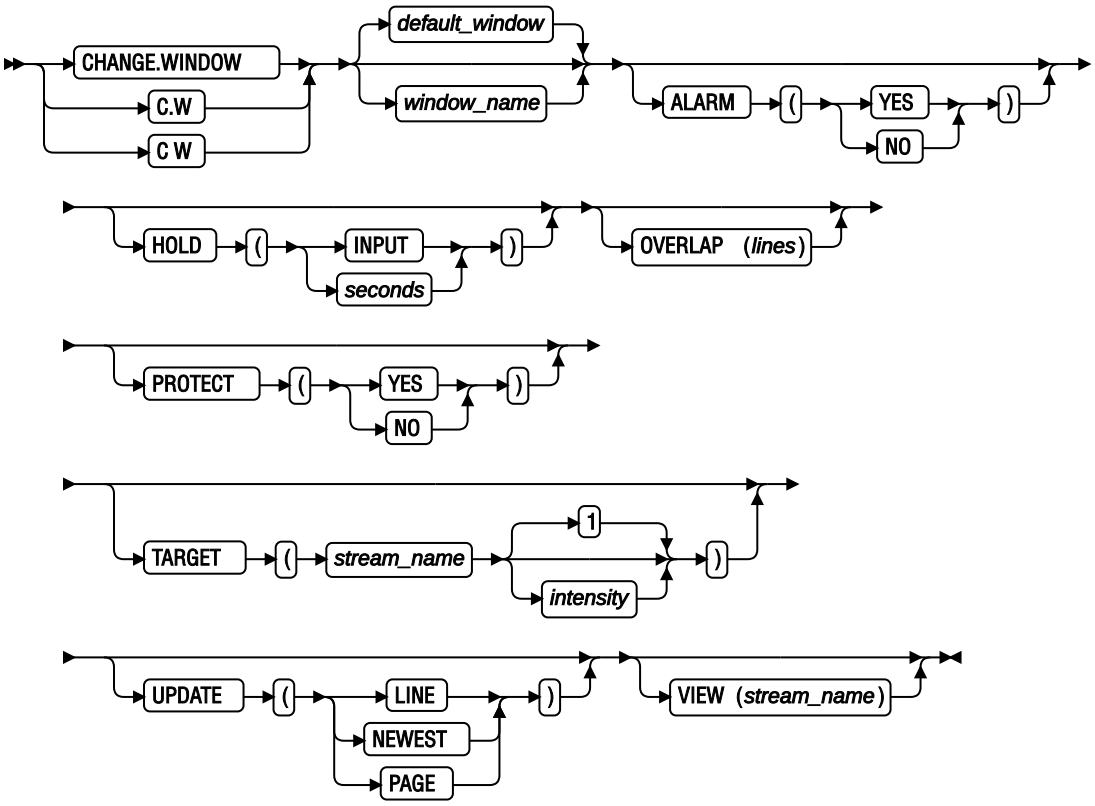
Set the terminal so that the keyboard will be locked for no more than 15 seconds and set the MAIN window as the default window.

```
change.terminal control(15) default(main)
```

CHANGE.WINDOW command

Use the CHANGE.WINDOW command to change the attributes of an existing window on the display screen. Use the QUERY.TERMINAL or QUERY.WINDOWS command to display the names and attributes of the currently defined windows.

CHANGE.WINDOW command syntax



CHANGE.WINDOW command operands

window_name
The name of the window whose attributes are to be changed.

ALARM(YES | NO)
specifies whether the terminal's audible alarm is to sound when the Session Manager places new information in the stream. If your terminal does not have an audible alarm, the Session Manager still accepts this operand. It has no way of knowing whether your terminal has an alarm.

HOLD(INPUT | *seconds*)
specifies how long the window (when unlocked) is to be held in place before it is scrolled toward the bottom of the stream.

INPUT
specifies that the window (when unlocked) be held in place until you supply input by pressing the Enter key or any program function (PF) key.

seconds
specifies that the window (when unlocked) be held in place the specified number of seconds before it is scrolled toward the bottom of the stream. *seconds* must be an integer from 0 to 999.

During the time the window is held in place, the keyboard remains locked. The keyboard unlocks when the time expires or when the window displays the bottom of the stream.

Note: The value specified on the CONTROL operand of the CHANGE.TERMINAL command overrides the value specified on this operand.

OVERLAP(*lines*)
specifies how many lines of the window's old position are to be repeated when the window scrolls to a new position.

lines must be an integer from 0 to 999. If you specify a value for *lines* that is greater than or equal to the number of lines in the window, the Session Manager adjusts *lines* to be the number of lines in the

window minus one. Thus, at least the bottom line of the window's old position appears at the top of the window's new position.

PROTECT(YES | NO)
specifies whether you can enter data in the window. You can enter data in an unprotected window only. If you try to enter data in a protected window, the keyboard locks.

TARGET(*stream_name intensity*)

stream_name
is the name of the stream that is to receive the information entered in the window.

intensity
specifies the brightness at which the information in the stream is to be displayed. The valid values are:

0
The information in the stream is not to be displayed. You can see the line that the information occupies, but the information itself is invisible.

1
The information is to be displayed at normal intensity.

2
The information is to be highlighted.

UPDATE(LINE | NEWEST | PAGE)
specifies how much new information must enter the stream before the Session Manager updates the window. The window scrolls only when it is unlocked.

LINE
specifies that the window scroll sequentially toward the bottom of the stream. Thus, all of the new information is displayed as the window scrolls over the stream. When the window is full, it scrolls forward (repeating the number of lines specified by the OVERLAP operand) and the new information again starts to fill up the window.

NEWEST
specifies that the window always display the newest information in the stream. When new information enters the stream, the window scrolls directly to the bottom of the stream. Some information in the stream might be skipped over. Thus, if a large amount of information is sent to the stream in a short period of time, only the last few lines (the number of lines in the window) are displayed.

PAGE
specifies that the window scroll sequentially over the stream when there are enough new lines of information (minus the number of overlap lines) to fill the window. The window does not scroll to display the new information until enough additional information (a "page" of information) enters the stream.

VIEW(*stream_name*)
The name of the stream the window is to display. Initially, the Session Manager places the window at the bottom of the stream and unlocks the window.

CHANGE.WINDOW command return codes

Table 67 on page 375 lists all the return codes of CHANGE.WINDOW command.

Table 67: CHANGE.WINDOW command return codes

Return codes	Meaning
0	Processing successful.
4	Syntax error in command.
8	Window not found.

CHANGE.WINDOW command examples

Example 1

Change the TEST window to display the SMOUT stream.

```
change.window test view(smout)
```

Example 2

Change the PASSWD window so that TSO/E passwords can be entered in non-display mode.

```
change.window passwd target(tsoin 0)
```

Example 3

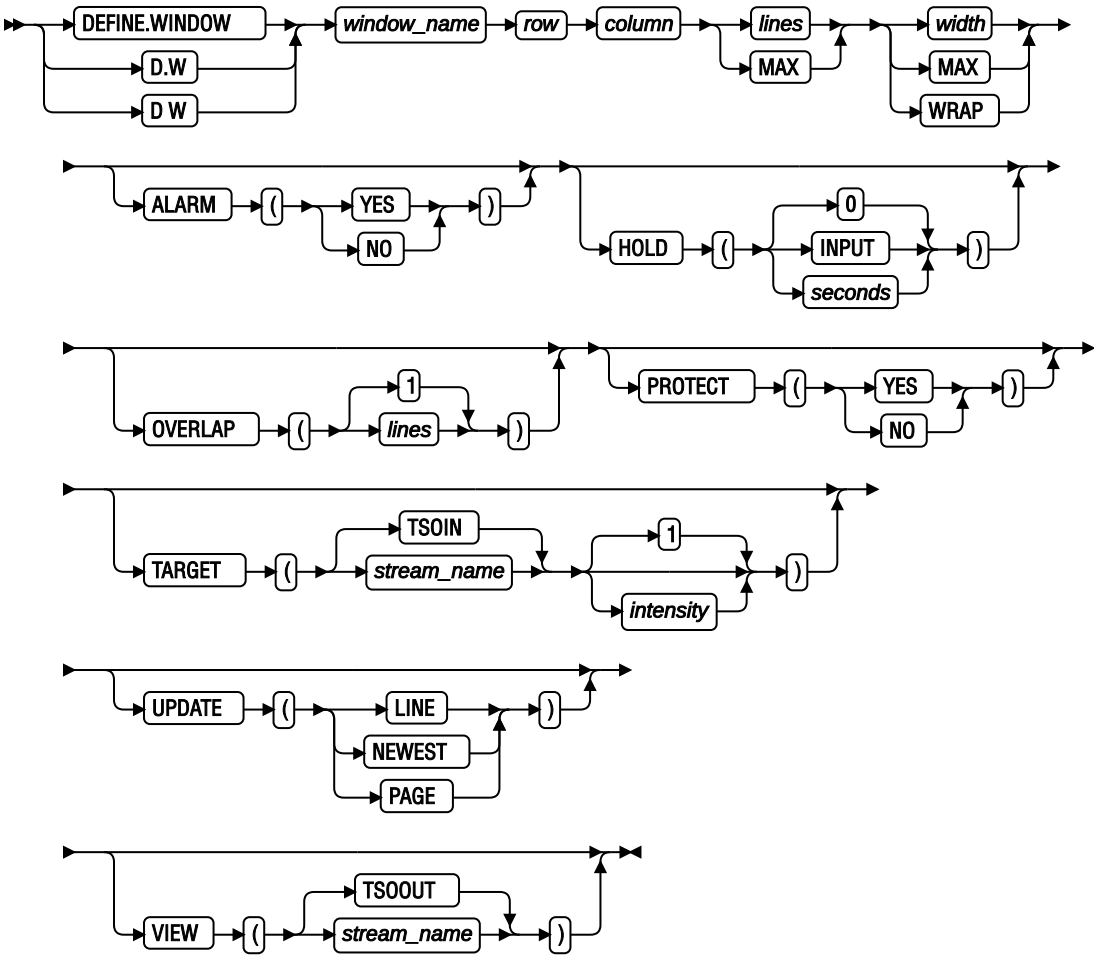
Change the default window so that input cannot be entered there.

```
change.window protect(yes)
```

DEFINE.WINDOW command

Use the DEFINE.WINDOW command to define a new window on the display screen.

DEFINE.WINDOW command syntax



DEFINE.WINDOW command operands

window_name
The name of the window being defined. The name must be 1 to 8 alphanumeric characters, with the first character alphabetic.

row
specifies which row of the display screen the top line of the window is to occupy. *row* must be an integer n or -n, where n can be any number from 1 to the number of rows on the display screen. An integer of -n is relative to the bottom of the screen. For example, a row value of -4 on a 24 line screen means that the top line of the window is to be row 21.

column
specifies which column of the display screen the left side of the window is to occupy. *column* must be an integer n or -n, where n can be any number from 1 to the number of columns on the screen. An integer of -n is relative to the right side of the display screen. For example, a column value of -4 on an 80 column screen means that the left side of the window is to be in column 77.

***lines* | MAX**
specifies the number of lines in the window. *lines* must be an integer *n* or the character string MAX. The value *n* can be any number from 1 to the number of lines on the display screen. MAX indicates that the window is to consist of the remaining lines on the display screen or until a line is encountered that has already been defined as part of another window.

***width* | MAX | WRAP**
specifies the number of character positions in each line of the window. *width* can be an integer n or the character string MAX or WRAP.

width
can be any number from 1 (or the number defined as the starting column) to the physical width of the display screen.

MAX
indicates that the width of the window should be determined by the number of character positions available in the first line of the window (those not used by another window).

WRAP
indicates that the width of the window is to start from the column value specified with this command and continue to either the beginning of the next window or to the last row and column of the screen. WRAP can only be used when *lines* is defined as 1.

Note: The first character position in a window is used as a terminal attribute byte and is protected. Therefore, a window defined with a width of 1 is useless.

ALARM(YES | NO)
specifies whether the terminal's audible alarm is to sound when the Session Manager scrolls the window to display new information in the stream. If your terminal does not have an audible alarm, the Session Manager still accepts this operand. It has no way of knowing whether your terminal has an alarm.

HOLD(INPUT | *seconds*)
specifies how long the window (when unlocked) is to be held in place before it is scrolled toward the bottom of the stream.

INPUT
specifies that the window (when unlocked) be held in place until you supply input by pressing the Enter key or any program function (PF) key.

seconds
specifies that the window (when unlocked) be held in place the specified number of seconds before it is scrolled toward the bottom of the stream. *seconds* must be an integer from 0 to 999.

During the time the window is held in place, the keyboard remains locked. The keyboard unlocks when the time expires or when the window displays the bottom of the stream.

Note: The value specified on the CONTROL operand of the CHANGE.TERMINAL command overrides the value specified on this operand.

OVERLAP(*lines*)
specifies how many lines of the window's old position are to be repeated when the window scrolls to the new position.

lines must be an integer from 0 to 999. If you specify a value for *lines* that is greater than or equal to the number of lines in the window, the Session Manager adjusts the value to be the number of lines in the window minus one. Thus, at least the bottom line of the window's old position always appears at the top of the window's new position.

PROTECT(YES | NO)
specifies whether you can enter data in the window. You can enter data in an unprotected window only. If you try to enter data in a protected window, the keyboard locks.

TARGET(*stream_name intensity*)

stream_name
The name of the stream that is to receive the information entered in the window.

intensity
specifies the brightness at which the information in the stream is to be displayed. The valid values are:

0
The information in the stream is not to be displayed. You can see the line that the information occupies, but the information itself is invisible.

1
The information is to be displayed at normal intensity.

2
The information is to be highlighted.

UPDATE(LINE | NEWEST | PAGE)
specifies how much new information must enter the stream before the Session Manager updates the window. The window scrolls only when it is unlocked.

LINE
specifies that the window scroll sequentially toward the bottom of the stream. Thus, all of the new information is displayed as the window scrolls over the stream. When the window is full, it scrolls forward (repeating the number of lines specified by the OVERLAP operand) and the new information again starts to fill up the window.

NEWEST
specifies that the window always display the newest information in the stream. When new information enters the stream, the window scrolls directly to the bottom of the stream. Some information in the stream might be skipped over. Thus, if a large amount of information is sent to the stream in a short period of time, only the last few lines (the number of lines in the window) are displayed.

PAGE
specifies that the window scroll sequentially over the stream when there are enough new lines of information (minus the number of overlap lines) to fill the window. The window does not scroll to display the new information until enough additional information (a "page" of information) enters the stream.

VIEW(*stream_name* | TSOOUT)
The name of the stream that the window is to display. Initially, the Session Manager places the window at the bottom of the stream and unlocks the window.

DEFINE.WINDOW command return codes

Table 68 on page 379 lists all the return codes of DEFINE.WINDOW command.

Table 68: DEFINE.WINDOW command return codes	
Return codes	**Meaning**
0	Processing successful.
4	Syntax error in command.
8	Window not found.

DEFINE.WINDOW command examples

Example 1

Note: The display screen for this example contains 24 lines and is 80 columns wide.

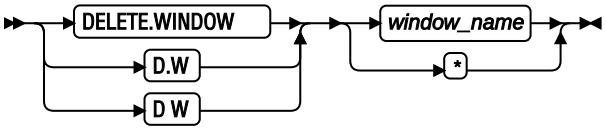
Create a screen layout having an output window occupying the top 22 lines of the screen with a character width of the entire screen and an input window that occupies the bottom two lines of the screen but is logically a single line. The output window is to display the TSOOUT stream and input window is to display the HEADER stream. All other attributes of the windows are to assume the default values.

```
define.window output 1 1 22 max
define.window input -2 1 1 wrap view(header)
```

DELETE.WINDOW command

Use the DELETE.WINDOW command to delete a window from the display screen.

DELETE.WINDOW command syntax



DELETE.WINDOW command operands

window_name
The name of the window to be deleted.

*****
specifies that all of the windows on the display screen are to be deleted. When all windows are deleted, press the CLEAR key before entering commands from the keyboard.

DELETE.WINDOW command return codes

Table 70 lists all the return codes of DELETE.WINDOW command.

Table 69: DELETE.WINDOW command return codes	
Return codes	**Meaning**
0	Processing successful.
4	Syntax error in command.
8	Window not found.

DELETE.WINDOW command examples

Example 1

Delete the TEST window.

```
delete.window test
```

Example 2

Delete all of the windows on the display screen.

```
delete.window *
```

END command

Use the END command to end Session Manager display support of your TSO/E session. The information in your streams is erased when you issue the END command. If, after issuing this command, you want to have Session Manager support again, you must reissue the TSO/E LOGON command.

END command syntax

FIND command

Use the FIND command to search for a specific text string in the stream that a window is displaying or to determine the number of the top line displayed in a window.

If the Session Manager finds the text string, it scrolls the window so that the line containing the text string is displayed on the top line of the window and the window is locked. If the text string is not found, the Session Manager places a message in the Session Manager output stream. If you issue the FIND command with a null text string (adjacent quote marks ''), the Session Manager searches for the previous text string.

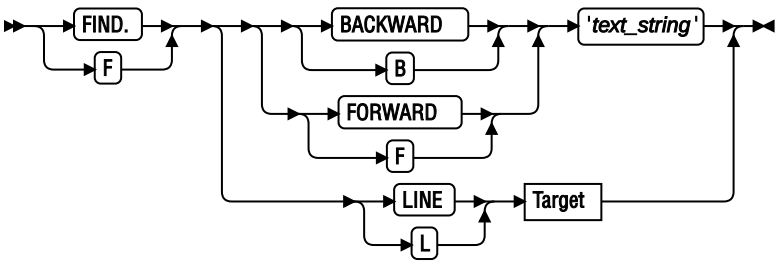
For the FIND.LINE command, the Session Manager writes the following message in the specified window:

```
ADF031I  window_name VIEWING LINE nnnnn
```

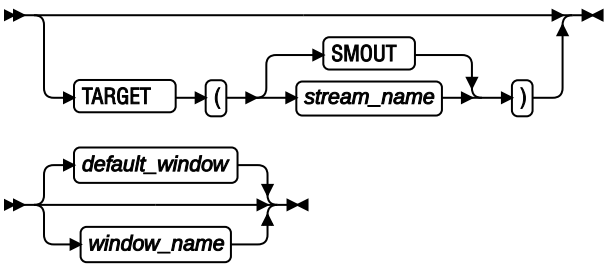
'nnnnn' is the top line number in the window.

The FIND.LINE command is useful for finding a line number for the SCROLL.ABSOLUTE command, for copying a range of lines using the SMCOPY command, or for locating a text string within a range of lines using the SMFIND command. The Session Manager does not lock the window when you use the FIND.LINE command.

FIND command syntax



Target



FIND command operands

BACKWARD
causes the Session Manager to search for the specified *text_string* from the top line in the window on the display screen backward toward the top of the stream. The stream searched is the one displayed in either the default window or the window specified on the command.

FORWARD
causes the Session Manager to search for the specified *text_string* from the current line on the display screen forward toward the bottom of the stream. The stream searched is the one displayed in either the default window or the window specified on the command.

LINE
causes the Session Manager to find the number of the top line in the default window or the specified window and writes a message identifying the line number in the specified stream.

text_string
The string of characters for which Session Manager is to search. If the *text_string* contains lowercase letters, blanks, commas, or parentheses, enclose it in single quotation marks. A single quotation mark in the *text_string* must be represented as two adjacent quotation marks.

If there are no blanks, commas, or parentheses in the *text_string*, you can omit the enclosing quotation marks. If you omit the enclosing quotation marks, however, the Session Manager translates the *text_string* to uppercase letters before beginning the search. When the FIND command is used in a CLIST, the Session Manager always stores the *text_string* in uppercase letters, even if it is enclosed in quotation marks.

If you specify a null *text_string*, the Session Manager uses the last *text_string* you entered as the string of characters to search for. A null *text_string* is defined as two adjacent quotation marks ('').

TARGET(*stream_name* | SMOUT)
The name of the stream that is to contain the message produced by the FIND.LINE command.

window_name
For FIND.BACKWARD and FIND.FORWARD, *window_name* is the name of the window whose stream is to be searched.

For FIND.LINE, *window_name* is the name of the window whose top line number is to be found.

FIND command return codes

Table 70 on page 381 lists all the return codes of FIND command.

Table 70: FIND command return codes

Return codes	Meaning
0	Processing successful.
4	Syntax error in command.
8	Text not found.

FIND command examples

Example 1

Assume that the default window is displaying the bottom of the TSOOUT stream. Find the last time the character string ‘link’ was issued.

```
find.backward 'link'
```

Example 2

Assume that the TEST window is displaying the top of the TSOOUT stream. Find the first time you edited a data set named ‘abc.asm’.

```
find.forward 'edit abc.asm' test
```

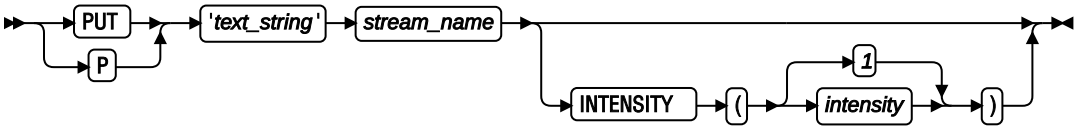
Now find the next occurrence of this same text string.

```
find.forward '' test
```

PUT command

Use the PUT command to place a text string in a Session Manager stream. If you place the text string in the TSOIN stream, it is sent to TSO/E to be executed as a TSO/E command. If you place the text string in the SMIN stream, it is interpreted as a Session Manager command. The length of the entire PUT command cannot exceed 512 characters.

PUT command syntax



PUT command operands

text_string
The string of characters to be placed in the stream. If the *text_string* contains lowercase letters, blanks, commas, parentheses, it must be enclosed in single quotation marks. A single quotation mark in the *text_string* must be represented as two adjacent quotation marks.

If there are no blanks, commas, or parentheses in the *text_string*, you can omit the enclosing quotation marks. If you omit the enclosing the enclosing quotation marks, however, the Session Manager translates the *text_string* to uppercase letters before beginning the search. When the PUT command is used in a CLIST, the Session Manager always stores the *text_string* in uppercase letters, even if it is enclosed in quotation marks.

stream_name
The name of the stream where the *text_string* is to be placed. The Session Manager places the *text_string* at the bottom of this stream.

INTENSITY(*intensity*)
specifies the brightness at which the *text_string* is to be displayed in the stream. The valid values are:

0
The *text_string* is not to be displayed. You can see the line that the *text_string* occupies, but the information itself is invisible.

1
The *text_string* is to be displayed at normal intensity.

2
The *text_string* is to be highlighted.

PUT command return codes

Table 71 on page 383 lists all the return codes of PUT command.

Table 71: PUT command return codes	
Return code	**Meaning**
0	Processing successful.
4	Syntax error in command.
8	Stream not found.

PUT command examples

Example 1

Place a comment in the TSOOUT stream and highlight it.

```
put 'this is a comment' tsoout intensity(2)
```

Example 2

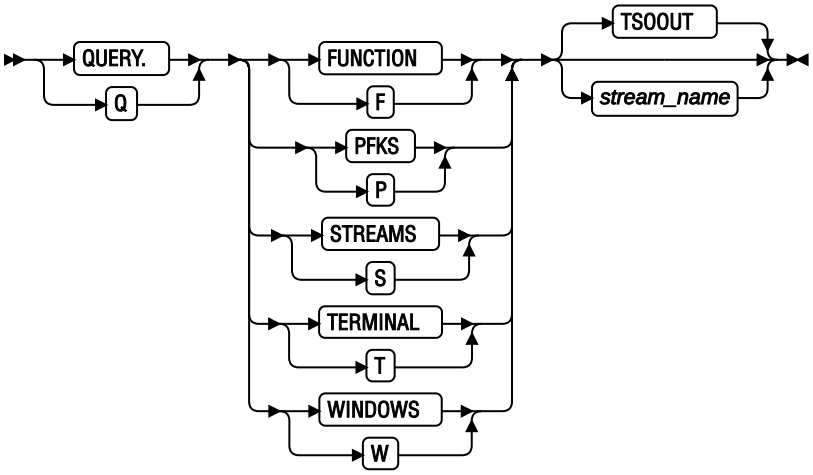
Use the PUT command in the *definition_text_string* of the CHANGE.PFK command. When pressed, PF3 is to issue the TSO/E TIME command and the Session Manager UNLOCK.NEWEST command. The commands are to be placed in the SMIN stream.

```
change.pfk 3 'put time tsoin;put 'unlock newest'
smin' smin
```

QUERY command

Use the QUERY command to display information about the Session Manager functions, program function (PF) keys, streams, terminal, and windows.

QUERY command syntax



QUERY command operands

FUNCTION
displays the following information for each currently defined session function:

- The name of the function
- The input, output, and copy streams for the function
- Whether the audible alarm is to sound when information enters the input and output streams
- The intensity at which the information in the output and copy streams is displayed.

PFKS
displays the following information for each currently defined program function (PF) key:

- The number of the PF key
- The name of the stream where the text_string is to be placed
- The identifier and delimiter characters for the PF keys
- The text_string used to define the key.

STREAMS
displays the following information for each currently defined stream:

- The name of the stream
- The numbers of the top and bottom lines in the stream
- The maximum size of the stream (in lines and bytes)
- The number of lines and bytes currently used by the stream
- The type of stream (input, output, or extra)
- Whether the audible alarm is to sound when information enters the stream.

TERMINAL
displays the following information about the terminal environment:

- The control setting for the keyboard indicating the maximum time the keyboard is to remain locked
- Whether the audible alarm is to sound when the keyboard unlocks
- The current number of windows defined on the display screen
- The maximum number of windows that can be defined
- The name of the default window
- The permanent location of the cursor
- The following information for each currently defined window:
  - The name of the window
  - The name of the stream that the window displays
  - Whether the window is locked
  - Whether you can enter data in the window
  - The name of the stream that is to receive the information entered in the window
  - The intensity at which the information in the stream is to be displayed
  - Whether the terminal's audible alarm is to sound when the Session Manager scrolls the window to display new information in the stream
  - How long the window (when unlocked) is to be held in place before it is scrolled toward the bottom of the stream
  - How many lines of the window's old position are to be repeated when the window scrolls to the new position
  - How much new information must enter the stream before the Session Manager updates the window.

WINDOWS
displays the following information for each currently defined window:

- The name of the window
- The starting location of the window on the display screen (in rows and columns)
- The size of the window (in lines and width)
- The name of the stream that the window displays
- The numbers of the top and bottom lines of the stream that the window is currently displaying
- The numbers of the top and bottom lines of the stream that the window was displaying when it was last unlocked. (These numbers are used when the UNLOCK.RESUME command is issued.)

- The numbers of the top and bottom lines of the newest information in the stream that the window is currently displaying.

stream_name
The name of the stream where the output from the command is to be placed. The output is in table format.

QUERY command return codes

Table 72 on page 385 lists all the return codes of QUERY command.

Table 72: QUERY command return codes.

Return codes	Meaning
0	Processing successful.
4	Syntax error in command.
8	Stream not found.

QUERY Command Examples

Example 1

Display the information for all session functions.

```
query.function
```

The output from the command is as follows:

```
FUNCTION     INPUT            OUTPUT           COPY
  NAME    STREAM  ALARM  STREAM  INT ALARM  STREAM  INT
TSO      TSOIN      N   TSOOUT    1    N   TSOOUT    2
SM       SMIN       N   SMOUT     2    Y    SMOUT    0
MSG       *NONE*        TSOOUT    2    Y     *NONE*
```

Example 2

Display the information for all streams defined.

```
query.streams
```

The output from the command is as follows:

```
 STREAM    LINE  RANGE   MAXIMUM  SIZE       USED
  NAME     LOW    HIGH  LINES     BYTES  LINES  BYTES  TYPE  ALARM
TSOIN        1       4    305      8192      4     82  INPUT   N
TSOOUT       1      47   4005    147456     47   4678  OUTPUT  N
EXTRA1       1       1    405     32768      1     38  OUTPUT  N
SMOUT        1      61    155      4096     61    424  OUTPUT  N
HEADER       1       9     55      1024      9    349  EXTRA   N
EXTRA3       1       2    105      1024      2     47  EXTRA   N
EXTRA2       1       1    105      1024      1     38  EXTRA   N
MESSAGE      1       1     55      1024      1     39  OUTPUT  Y
SMIN         1      59    305      8192     59   6192  INPUT   N
QUERY COMPLETE
```

Example 3

Display all information related to the terminal.

```
query.terminal
```

The output from the command is as follows:

```
KEYBOARD   CNTL    ALARM
             15      N
WINDOWS    CURRENT # MAXIMUM #       DEFAULT WINDOW    CURSOR POSITION
             11          25            MAIN           ENTRY      1    1
  NAME    VIEW  LOCKED  PROT  TARGET  INTENSITY  ALARM  HOLD  OVERLAP  UPDATE
LINE    HEADER    Y       Y   TSOIN       1         N      0        0    N
STITLE  HEADER    Y       Y   TSOIN       1         N      0        0    N
SVALUE  EXTRA3    N       Y   EXTRA3      1         N      0        0    N
LTITLE  HEADER    Y       Y   TSOIN       1         N      0        0    N
LVALUE  HEADER    Y       Y   TSOIN       1         N      0        0    N
VLINE   HEADER    Y       Y   TSOIN       1         N      0        0    N
PASSWD  SMOUT     N       N   TSOIN       0         N      0        0    N
CURRENT TSOOUT    N       N   TSOIN       1         N      0        0    N
TENTRY  HEADER    Y       Y   TSOIN       1         N      0        0    N
ENTRY   HEADER    Y       N   TSOIN       1         N      0        0    N
MAIN    TSOOUT    N       N   TSOIN       1         N      I        9    L
QUERY COMPLETE
```

Example 4

Display the information for all windows defined:

```
query.windows
```

The output from the command is as follows:

```
                                        PRESENT       RESUME       NEWEST
                                VIEWING  TOP  BOTTOM  TOP  BOTTOM TOP  BOTTOM
 WINDOW  ROW COL LINES WIDTH  STREAM   LINE  LINE   LINE  LINE   LINE  LINE
LINE      20   1     1    80  HEADER      2     2      1     1     10    10
STITLE    21  63     1    12  HEADER      6     6      1     1     10    10
SVALUE    21  75     1     6  EXTRA3      2     2      1     1      2     2
LTITLE    22  63     1     9  HEADER      9     9      1     1     10    10
LVALUE    22  72     1     9  HEADER      7     7      1     1     10    10
VLINE     24  41     1     2  HEADER      4     4      1     1     10    10
PASSWD    24  43     1    38  SMOUT      57    57      1     1     10    10
CURRENT   21   1     2    62  TSOOUT     36    37     36    37     46    47
TENTRY    23   1     1     5  HEADER      3     3      1     1     10    10
ENTRY     23   6     1   114  HEADER     10    10      1     1     10    10
MAIN       1   1    19    80  TSOOUT      1    19      1    19     32    50
QUERY COMPLETE
```

RESET command

Use the RESET command to restart your Session Manager display environment. For example, if you accidentally deleted any of the windows on your display screen, use RESET to get the default display screen back. This command removes the entries from all of the stacks and re-executes the commands that created the default environment. The RESET command causes the HEADER stream to be cleared and then redefined containing only those lines needed in the default environment. In addition, the default scroll amount is placed in the EXTRA3 stream, thereby resetting the default scroll amount on your display screen. None of the other streams are altered.

The RESET command should not be followed by any other Session Manager command on the same line. A command entered on the same line executes before the RESET command can reestablish the default screen layout.

RESET command syntax

RESET command return codes

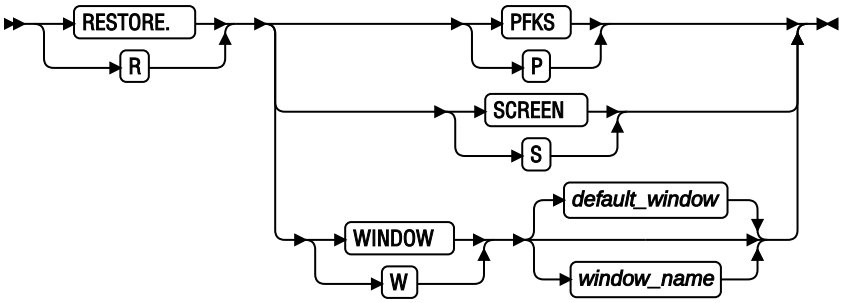
Table 73 on page 387 lists all the return codes of RESET command.

Table 73: RESET command return codes	
Return codes	**Meaning**
0	Processing successful.
4	Syntax error in command.
8	Window not found.

RESTORE command

Use the RESTORE command to restore the definitions of the program function (PF) keys, screen layout, or windows previously saved through the SAVE command. If only one set of definitions exists on the stack, it is not removed. If more than one set of definitions has been saved, you must issue the RESTORE command as many times as you issued the SAVE command to get to the definitions you want.

RESTORE command syntax



RESTORE command operands

PFKS
specifies that the Session Manager is to restore the program function (PF) key definitions.

SCREEN
specifies that the Session Manager is to restore the screen layout. The following items are included in each screen stack element:

- A description of the screen layout
- The location of the cursor
- The value indicating how long the keyboard is to remain locked while a command is executing (as set using the CHANGE.TERMINAL command)
- The name of the default window
- The name and attributes of each window.

WINDOW
specifies that the Session Manager is to restore the window definitions. Each window description element contains the following information:

- The audible alarm setting for the window (ALARM)
- The amount of time the window (when unlocked) is held in place before it is scrolled toward the bottom of the stream (HOLD)
- The number of lines from the window's old position that are to be repeated when the window scrolls (OVERLAP)
- Whether you can enter data in the window (PROTECT)
- The name of the stream that is to receive the information typed in the window and the intensity at which the information is to be displayed (TARGET)
- How often the window is to scroll over the new information in the stream (UPDATE)
- The name of the stream the window is displaying (VIEW)

- The numbers of the top and bottom lines in the stream that the window is currently displaying
- Whether the window is locked or unlocked

The location and size of the window are not restored.

window_name
The name of the window whose description is to be restored.

RESTORE command return codes

Table 74 on page 388 lists all the return codes of RESTORE command.

Table 74: RESTORE command return codes

Return codes	Meaning
0	Processing successful.
4	Syntax error in command.
8	Stream or window not found.

RESTORE command examples

Example 1

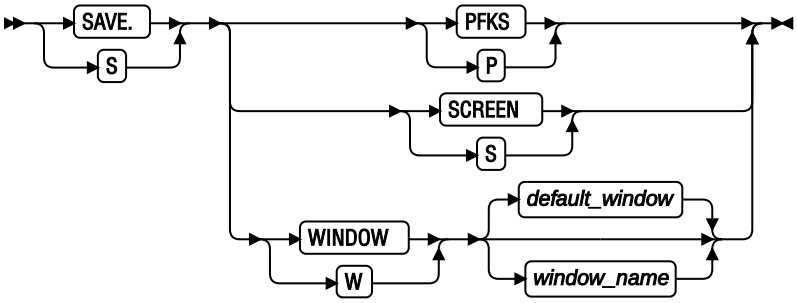
Restore the definition of the TEST window that was previously saved through the SAVE command.

```
restore.window test
```

SAVE command

Use the SAVE command to save the current definitions of program function (PF) keys, windows, and the screen layout. Later, you can restore these same definitions by using the RESTORE command.

SAVE command syntax



SAVE command operands

PFKS
specifies that all current PF key definitions are to be saved as the top element of the PF key stack.

SCREEN
specifies that the current screen layout is to be saved as the top element on the screen stack. The following items are saved for each screen stack element:

- A description of the screen layout
- The location of the cursor
- The value indicating how long the keyboard is to remain locked while a command is executing (as set using the CHANGE.TERMINAL command)
- The name of the default window

- The name and attributes of each window.

WINDOW
requests that the definitions for the default window or the window specified on the command be saved as the top element of the window stack. Each window description element contains the following information:

- The audible alarm setting for the window (ALARM)
- The amount of time the window (when unlocked) is held in place before it is scrolled toward the bottom of the stream (HOLD)
- The number of lines from the window's old position that are to be repeated when the window scrolls (OVERLAP)
- Whether you can enter data in the window (PROTECT)
- The name of the stream that is to receive the information typed in the window and the intensity at which the information is to be displayed (TARGET)
- How often the window is to scroll over the new information in the stream (UPDATE)
- The name of the stream the window is displaying (VIEW)
- The numbers of the top and bottom lines in the stream that the window is currently displaying
- Whether the window is locked or unlocked

The location and size of the window are not saved.

window_name
The name of the window whose description is to be saved.

SAVE command return codes

Table 75 on page 389 lists all the return codes of SAVE command.

Table 75: SAVE command return codes

Return codes	Meaning
0	Processing successful.
4	Syntax error in command.
8	Window not found.

SAVE command examples

Example 1

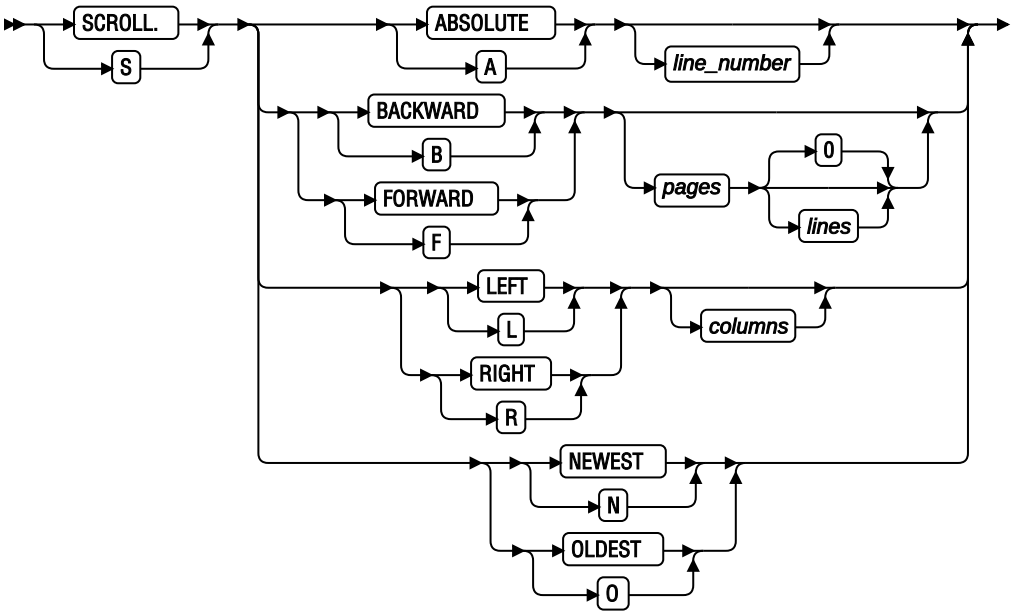
Save the definition of the TEST window on the window stack.

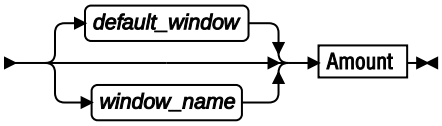
```
save.window test
```

SCROLL command

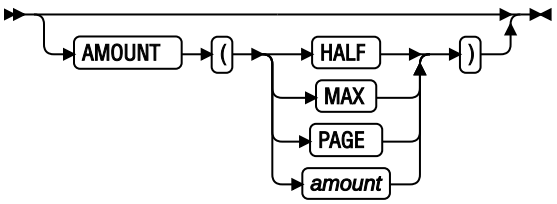
Use the SCROLL command to move a window over a stream. After the Session Manager moves the window, the window is locked in position. You can then move the window by using another scroll command or you can unlock the window by using the UNLOCK command.

SCROLL command syntax





Amount



SCROLL command operands

ABSOLUTE
specifies that the Session Manager is to scroll the window so that the identified *line_number* is the top line in the window. Use the QUERY, SMFIND, or FIND.LINE commands to find specific line numbers.

line_number
The number of the line you want to appear at the top of the window. If you enter a value for *line_number* that is 0 or less, the Session Manager sets *line_number* to 1. If you enter a value for *line_number* that is greater than the highest line number in the stream, the Session Manager sets *line_number* to the highest line number in the stream.

BACKWARD
specifies that the Session Manager is to scroll the window backward toward the top of the stream.

pages
specifies how many pages to scroll the window. (A page is defined as the number of lines in the window.)

If the AMOUNT keyword is entered, the default is 0. If the AMOUNT keyword is not entered, the default is 1.

If you specify a value that would cause the window to scroll beyond the top or bottom of the stream, the Session Manager adjusts the value to place the window at the top or bottom (depending on the direction of the scrolling).

lines
specifies how many lines to scroll the window.

If you specify a value that would cause the window to scroll beyond the top or bottom of the stream, the Session Manager adjusts the value to place the window at the top or bottom (depending on the direction of the scrolling).

FORWARD
specifies that the Session Manager is to scroll the window forward toward the bottom of the stream.

LEFT
specifies that the Session Manager is to scroll the window toward the left side of the stream. The limit for scrolling left is column 1 of the stream.

columns
specifies the number of columns to scroll the window.

If the AMOUNT keyword is entered, the default is 0. If the AMOUNT keyword is not entered, the default is 40.

If you specify a value that would cause the window to scroll beyond the left side of the stream, the Session Manager adjusts the value to place the window at column 1 of the stream. Values that would cause the window to scroll beyond column 32768 are adjusted to place the window at column 32,768.

RIGHT
specifies that the Session Manager is to scroll the window toward the right side of the stream. The limit for scrolling right is 32,768 column positions.

NEWEST
specifies that the Session Manager is to scroll the window forward to the bottom of the stream.

OLDEST
specifies that the Session Manager is to scroll the window backward to the top of the stream.

window_name
The name of the window to be scrolled.

AMOUNT
The amount the window is to be scrolled. AMOUNT can be specified instead of or in addition to, the operands *columns*, *lines*, or *pages*. If you enter a value for one of the preceding operands and a value for AMOUNT, the Session Manager sums the two values and scrolls the window the resulting amount. The valid AMOUNT values are:

HALF
specifies that the window is to be scrolled half a page. (For forward or backward scrolling, a page is defined as the number of lines in the window. For right or left scrolling, a page is defined as the number of columns in the window.)

MAX
specifies that the window is to be scrolled the maximum amount. For forward scrolling, MAX indicates to scroll to the bottom of the stream (equivalent to the SCROLL.NEWEST command). For backward scrolling, MAX indicates to scroll to the top of the stream (equivalent to the SCROLL.OLDEST command).

PAGE
specifies that the window is to be scrolled a full page.

amount
specifies the number of lines or columns to scroll the window.

SCROLL command return codes

Table 76 on page 392 lists all the return codes of SCROLL command.

Table 76: SCROLL command return codes	
Return codes	**Meaning**
0	Processing successful.
4	Syntax error in command.
8	Window not found.

SCROLL command examples

Example 1

Scroll the default window to the oldest information.

```
scroll.oldest
```

or

```
scroll.backward amount(max)
```

Example 2

Scroll the TEST window forward one page.

```
scroll.forward test
```

or

```
scroll.forward test amount(page)
```

Example 3

Scroll the SAMPLE window backward 20 lines.

```
scroll.backward 0 20 sample
```

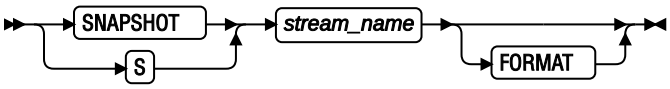
or

```
scroll.backward sample amount(20)
```

SNAPSHOT command

Use the SNAPSHOT command to copy a display screen of information into a stream. You can then use the SMCOPY command to print the stream or copy it into a data set.

SNAPSHOT command syntax



SNAPSHOT command operands

stream_name
The name of the stream where the information is to go.

FORMAT
specifies that carriage control information is to be included in the copy of the information for printing on a system printer. Highlighted lines on the screen appear darker in the printed copy.

Note: If the copied information contains the carriage control characters, you must use the PREFORMAT operand of the SMCOPY command when printing it.

SHAPSHOT command return codes

Table 77 on page 393 lists all the return codes of SHAPSHOT command.

Table 77: SHAPSHOT command return codes

Return codes	Meaning
0	Processing successful.
4	Syntax error in command.
8	Stream not found.

SHAPSHOT command examples

Example 1

Place a copy of the display screen in the EXTRA1 stream.

```
snapshot extra1 format
```

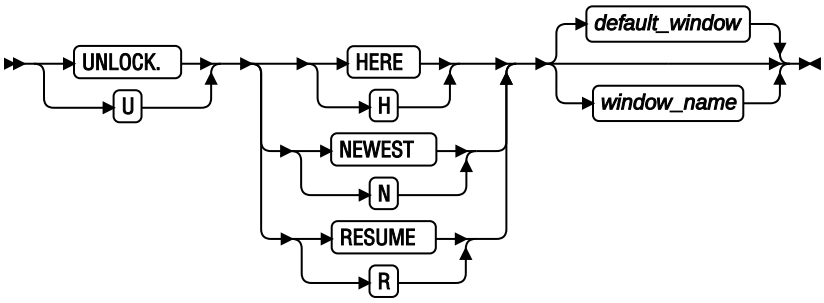
Print the stream, using the SMCOPY command.

```
smcopy fromstream(extra1) preformat
```

UNLOCK command

Use the UNLOCK command to unlock a window.

UNLOCK command syntax



UNLOCK command operands

HERE
causes the Session Manager to unlock the specified window at its current position.

NEWEST
causes the Session Manager to display the newest information in the stream, then unlocks it.

RESUME
causes the Session Manager to display the information the window was viewing before being locked, then unlocks the window.

window_name
The name of the window to be unlocked.

UNLOCK command return codes

Table 78 on page 394 lists all the return codes of UNLOCK command.

Table 78: UNLOCK command return codes

Return code	Meaning
0	Processing successful.
4	Syntax error in command.
8	Window not found.

UNLOCK command examples

Example 1

Move the default window to the bottom of the stream it is displaying and unlock it there.

```
unlock.newest
```

Example 2

Unlock the SAMPLE window at its current position.

```
unlock.here sample
```

Appendix A. Accessibility

Accessible publications for this product are offered through IBM Knowledge Center (www.ibm.com/support/knowledgecenter/SSLTBW/welcome).

If you experience difficulty with the accessibility of any z/OS information, send a detailed email message to mhvrcfs@us.ibm.com.

Accessibility features

Accessibility features help users who have physical disabilities such as restricted mobility or limited vision use software products successfully. The accessibility features in z/OS can help users do the following tasks:

- Run assistive technology such as screen readers and screen magnifier software.
- Operate specific or equivalent features by using the keyboard.
- Customize display attributes such as color, contrast, and font size.

Consult assistive technologies

Assistive technology products such as screen readers function with the user interfaces found in z/OS. Consult the product information for the specific assistive technology product that is used to access z/OS interfaces.

Keyboard navigation of the user interface

You can access z/OS user interfaces with TSO/E or ISPF. The following information describes how to use TSO/E and ISPF, including the use of keyboard shortcuts and function keys (PF keys). Each guide includes the default settings for the PF keys.

- *z/OS TSO/E Primer*
- *z/OS TSO/E User's Guide*
- *z/OS ISPF User's Guide Vol I*

Dotted decimal syntax diagrams

Syntax diagrams are provided in dotted decimal format for users who access IBM Knowledge Center with a screen reader. In dotted decimal format, each syntax element is written on a separate line. If two or more syntax elements are always present together (or always absent together), they can appear on the same line because they are considered a single compound syntax element.

Each line starts with a dotted decimal number; for example, 3 or `3.1` or `3.1.1`. To hear these numbers correctly, make sure that the screen reader is set to read out punctuation. All the syntax elements that have the same dotted decimal number (for example, all the syntax elements that have the number `3.1`) are mutually exclusive alternatives. If you hear the lines `3.1 USERID` and `3.1 SYSTEMID`, your syntax can include either `USERID` or `SYSTEMID`, but not both.

The dotted decimal numbering level denotes the level of nesting. For example, if a syntax element with dotted decimal number 3 is followed by a series of syntax elements with dotted decimal number 3.1, all the syntax elements numbered 3.1 are subordinate to the syntax element numbered 3.

Certain words and symbols are used next to the dotted decimal numbers to add information about the syntax elements. Occasionally, these words and symbols might occur at the beginning of the element itself. For ease of identification, if the word or symbol is a part of the syntax element, it is preceded by the backslash (\) character. The * symbol is placed next to a dotted decimal number to indicate that the syntax element repeats. For example, syntax element *FILE with dotted decimal number 3 is given the format 3 * FILE. Format 3* FILE indicates that syntax element FILE repeats. Format 3* * FILE indicates that syntax element * FILE repeats.

Characters such as commas, which are used to separate a string of syntax elements, are shown in the syntax just before the items they separate. These characters can appear on the same line as each item, or on a separate line with the same dotted decimal number as the relevant items. The line can also show another symbol to provide information about the syntax elements. For example, the lines 5.1*, 5.1 LASTRUN, and 5.1 DELETE mean that if you use more than one of the LASTRUN and DELETE syntax elements, the elements must be separated by a comma. If no separator is given, assume that you use a blank to separate each syntax element.

If a syntax element is preceded by the % symbol, it indicates a reference that is defined elsewhere. The string that follows the % symbol is the name of a syntax fragment rather than a literal. For example, the line 2.1 %OP1 means that you must refer to separate syntax fragment OP1.

The following symbols are used next to the dotted decimal numbers.

? indicates an optional syntax element
The question mark (?) symbol indicates an optional syntax element. A dotted decimal number followed by the question mark symbol (?) indicates that all the syntax elements with a corresponding dotted decimal number, and any subordinate syntax elements, are optional. If there is only one syntax element with a dotted decimal number, the ? symbol is displayed on the same line as the syntax element, (for example 5? NOTIFY). If there is more than one syntax element with a dotted decimal number, the ? symbol is displayed on a line by itself, followed by the syntax elements that are optional. For example, if you hear the lines 5 ?, 5 NOTIFY, and 5 UPDATE, you know that the syntax elements NOTIFY and UPDATE are optional. That is, you can choose one or none of them. The ? symbol is equivalent to a bypass line in a railroad diagram.

! indicates a default syntax element
The exclamation mark (!) symbol indicates a default syntax element. A dotted decimal number followed by the ! symbol and a syntax element indicate that the syntax element is the default option for all syntax elements that share the same dotted decimal number. Only one of the syntax elements that share the dotted decimal number can specify the ! symbol. For example, if you hear the lines 2? FILE, 2.1! (KEEP), and 2.1 (DELETE), you know that (KEEP) is the default option for the FILE keyword. In the example, if you include the FILE keyword, but do not specify an option, the default option KEEP is applied. A default option also applies to the next higher dotted decimal number. In this example, if the FILE keyword is omitted, the default FILE(KEEP) is used. However, if you hear the lines 2? FILE, 2.1, 2.1.1! (KEEP), and 2.1.1 (DELETE), the default option KEEP applies only to the next higher dotted decimal number, 2.1 (which does not have an associated keyword), and does not apply to 2? FILE. Nothing is used if the keyword FILE is omitted.

*** indicates an optional syntax element that is repeatable**
The asterisk or glyph (*) symbol indicates a syntax element that can be repeated zero or more times. A dotted decimal number followed by the * symbol indicates that this syntax element can be used zero or more times; that is, it is optional and can be repeated. For example, if you hear the line 5.1* data area, you know that you can include one data area, more than one data area, or no data area. If you hear the lines 3* , 3 HOST, 3 STATE, you know that you can include HOST, STATE, both together, or nothing.

Notes:

1. If a dotted decimal number has an asterisk (*) next to it and there is only one item with that dotted decimal number, you can repeat that same item more than once.
2. If a dotted decimal number has an asterisk next to it and several items have that dotted decimal number, you can use more than one item from the list, but you cannot use the items more than once each. In the previous example, you can write HOST STATE, but you cannot write HOST HOST.

3. The * symbol is equivalent to a loopback line in a railroad syntax diagram.

+ indicates a syntax element that must be included

The plus (+) symbol indicates a syntax element that must be included at least once. A dotted decimal number followed by the + symbol indicates that the syntax element must be included one or more times. That is, it must be included at least once and can be repeated. For example, if you hear the line `6.1+ data area`, you must include at least one data area. If you hear the lines `2+, 2 HOST,` and `2 STATE`, you know that you must include `HOST, STATE,` or both. Similar to the * symbol, the + symbol can repeat a particular item if it is the only item with that dotted decimal number. The + symbol, like the * symbol, is equivalent to a loopback line in a railroad syntax diagram.

Notices

This information was developed for products and services that are offered in the USA or elsewhere.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
United States of America

For license inquiries regarding double-byte character set (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

Intellectual Property Licensing
Legal and Intellectual Property Law
IBM Japan Ltd.
19-21, Nihonbashi-Hakozakicho, Chuo-ku
Tokyo 103-8510, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

This information could include missing, incorrect, or broken hyperlinks. Hyperlinks are maintained in only the HTML plug-in output for the Knowledge Centers. Use of hyperlinks in other output formats of this information is at your own risk.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Site Counsel
2455 South Road

Poughkeepsie, NY 12601-5400
USA

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. The sample programs are provided "AS IS", without warranty of any kind. IBM shall not be liable for any damages arising out of your use of the sample programs.

Terms and conditions for product documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Applicability

These terms and conditions are in addition to any terms of use for the IBM website.

Personal use

You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative work of these publications, or any portion thereof, without the express consent of IBM.

Commercial use

You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or

reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of IBM.

Rights

Except as expressly granted in this permission, no other permissions, licenses or rights are granted, either express or implied, to the publications or any information, data, software or other intellectual property contained therein.

IBM reserves the right to withdraw the permissions granted herein whenever, in its discretion, the use of the publications is detrimental to its interest or, as determined by IBM, the above instructions are not being properly followed.

You may not download, export or re-export this information except in full compliance with all applicable laws and regulations, including all United States export laws and regulations.

IBM MAKES NO GUARANTEE ABOUT THE CONTENT OF THESE PUBLICATIONS. THE PUBLICATIONS ARE PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, AND FITNESS FOR A PARTICULAR PURPOSE.

IBM Online Privacy Statement

IBM Software products, including software as a service solutions, ("Software Offerings") may use cookies or other technologies to collect product usage information, to help improve the end user experience, to tailor interactions with the end user, or for other purposes. In many cases no personally identifiable information is collected by the Software Offerings. Some of our Software Offerings can help enable you to collect personally identifiable information. If this Software Offering uses cookies to collect personally identifiable information, specific information about this offering's use of cookies is set forth below.

Depending upon the configurations deployed, this Software Offering may use session cookies that collect each user's name, email address, phone number, or other personally identifiable information for purposes of enhanced user usability and single sign-on configuration. These cookies can be disabled, but disabling them will also eliminate the functionality they enable.

If the configurations deployed for this Software Offering provide you as customer the ability to collect personally identifiable information from end users via cookies and other technologies, you should seek your own legal advice about any laws applicable to such data collection, including any requirements for notice and consent.

For more information about the use of various technologies, including cookies, for these purposes, see IBM's Privacy Policy at ibm.com/privacy and IBM's Online Privacy Statement at ibm.com/privacy/details in the section entitled "Cookies, Web Beacons and Other Technologies," and the "IBM Software Products and Software-as-a-Service Privacy Statement" at ibm.com/software/info/product-privacy.

Policy for unsupported hardware

Various z/OS elements, such as DFSMS, JES2, JES3, and MVS, contain code that supports specific hardware servers or devices. In some cases, this device-related element support remains in the product even after the hardware devices pass their announced End of Service date. z/OS may continue to service element code; however, it will not provide service related to unsupported hardware devices. Software problems related to these devices will not be accepted for service, and current service activity will cease if a problem is determined to be associated with out-of-support devices. In such cases, fixes will not be issued.

Minimum supported hardware

The minimum supported hardware for z/OS releases identified in z/OS announcements can subsequently change when service for particular servers or devices is withdrawn. Likewise, the levels of other software products supported on a particular release of z/OS are subject to the service support lifecycle of those products. Therefore, z/OS and its product publications (for example, panels, samples, messages, and product documentation) can include references to hardware and software that is no longer supported.

- For information about software support lifecycle, see: IBM Lifecycle Support for z/OS (www.ibm.com/software/support/systemsz/lifecycle)
- For information about currently-supported IBM hardware, contact your IBM representative.

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at Copyright and Trademark information (www.ibm.com/legal/copytrade.shtml).

Index

Special Characters

Numerics

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

SA32-0975-30