SC28-1307-1
File No. S370-39

Program Product

TSO Extensions Command Language Reference

TSO Extensions (TSO/E) 5665-285

IBM

Second Edition (January 1986)

This is a major revision of, and obsoletes, SC28-1307-0 and Technical Newsletter SN28-1030. See the Summary of Amendments following the Contents for a summary of the changes made to this manual. Technical changes or additions to the text and illustrations are indicated by a vertical line to the left of the change.

This edition applies to TSO Extensions (TSO/E) Release 2, Program Number 5665-285, until otherwise indicated in new editions or Technical Newsletters. Changes are made periodically to the information herein. Before using this publication in connection with the operation of IBM systems, consult the latest *IBM System Bibliography*, GC20-0001, for the editions that are applicable and current.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM program product in this publication is not intended to state or imply that only IBM's program product may be used. Any functionally equivalent program may be used instead.

Publications are not stocked at the address given below. Requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, Information Development, Department D58, Building 921-2, PO Box 390, Poughkeepsie, N.Y. 12602. IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Preface

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

If you are not familiar with TSO, you should first consult *TSO Extensions User's Guide*. If you have little or no knowledge of the use of TSO commands, *TSO Extensions User's Guide* provides the prerequisite information for using this book. The guide explains how to:

- Enter and execute commands
- Name and create specific types of data sets
- Rename, list, copy, protect, free, and delete data sets
- Compile and/or assemble code
- Link edit object modules
- Load and execute load modules.

How This Book is Organized

The major sections in this book are:

Basic Information for Using TSO, which contains general information necessary to use TSO commands. The section describes the syntax notation in the diagrams that accompany each command, positional and keyword operands, delimiters, line continuation, comments, and subcommands.

The Commands and Subcommands, which explain the syntax and function of each command, its operands, and its subcommands. Examples are included.

The CLIST Statements, also known as command procedures. Information on how to write CLISTs is in *CLISTs: Implementation and Reference*.

Quick Reference Guide to Commands and Statements, which contains a list of TSO commands and CLIST statements.

This book presents commands in alphabetical order. The subcommands are alphabetized under their commands. For example, all EDIT subcommands are alphabetized under the EDIT command. Included in the list of commands are CLIST statements.

The alphabetized guide to each TSO command also includes information on how that command executes in the background (batch processing), that is, independent of the terminal.

This reference book is for users of TSO Extensions running under (a) MVS/370, or (b) MVS/Extended Architecture (MVS/XA).

- If information or a command pertains to MVS/XA only, you will see next to that information or command: **MVS/XA Only**.
- If an operand pertains to MVS/XA only, you will see a footnote.

Program Products

This book refers to the following IBM program products:

Access Method Services Cryptographic Option, 5740-AM8
Cryptographic Unit Support, 5740-XY6
Data Facility Product (DFP), 5665-295 (**MVS/370**)
Data Facility Product (DFP), 5665-284 (**MVS/XA**)
Programmed Cryptographic Facility, 5740-XY5

Interactive System Productivity Facility (ISPF), 5668-960 or 5665-319
Interactive System Productivity Facility/Program Development Facility (ISPF/PDF), 5665-268 or 5665-317

Resource Access Control Facility (RACF), 5740-XXH

TSO Extensions (TSO/E), 5665-285

Assembler H Version 2, 5668-962
TSO Assembler Prompter, 5734-CP2
OS/VS COBOL Release 2.4, 5740-CB1
TSO COBOL Prompter, 5734-CP1
PL/I OS Optimizing Compiler, 5734-PL1
PL/I OS Checkout Compiler, 5734-PL2
VS FORTRAN, 5748-FO3
TSO FORTRAN Prompter, 5734-CP3
VSBASIC, 5748-XX1

Related Books

The following are the publications mentioned in this book.

MVS/370

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

MVS Data Management Services, GC26-4058

MVS Access Method Services Reference for the Integrated Catalog Facility, GC26-4051

MVS Access Method Services Reference for VSAM Catalogs, GC26-4059

OS/VS2 Assembler Language, GC33-4010

IBM 3800 Printing Subsystem Programmer's Guide, GC26-3846

IBM System/370 Reference Summary, GX20-1850

For the following books, see *MVS/System Product Version 1 General Information Manual*, GC28-1025, for the order numbers that correspond to the level you are using:

MVS JCL

OS/VS2 Message Library: System Messages

OS/VS2 System Programming Library: Job Management

OS/VS2 System Programming Library: Supervisor

MVS/XA

MVS/Extended Architecture Message Library: TSO Terminal Messages, GC38-1046

MVS/Extended Architecture TSO Terminal Monitor Program and Service Routines Logic, SY28-0650, as amended by Supplement LD23-0262

MVS/Extended Architecture Access Method Services Reference for the Integrated Catalog Facility, GC26-4019

MVS/Extended Architecture Access Method Services Reference for VSAM Catalogs, GC26-4075

MVS/Extended Architecture Data Administration Guide, GC26-4013

MVS/Extended Architecture Linkage Editor and Loader User's Guide, GC26-4011

MVS/Extended Architecture System Generation Reference, GC26-4009

For the following books, see *MVS/System Product Version 2 General Information Manual*, GC28-1118, for the order numbers that correspond to the level you are using:

MVS/Extended Architecture JCL

MVS/Extended Architecture Message Library: System Messages

MVS/Extended Architecture System Programming Library: System Macros and Facilities, Volume 1

MVS/Extended Architecture System Programming Library: System Macros and Facilities, Volume 2

MVS/Extended Architecture System Programming Library: System Modifications

MVS/Extended Architecture System Programming Library: 31-Bit Addressing

MVS/370 and MVS/XA

TSO Extensions Command Language Reference Summary, GX23-0015

TSO Extensions User's Guide, SC28-1333

TSO Extensions CLISTs: Implementation and Reference, SC28-1304

TSO Extensions Guide to Writing a Terminal Monitor Program or a Command Processor, SC28-1136

TSO Extensions Session Manager Program Reference, SC28-1306

TSO Extensions Terminal Messages, GC28-1310

System Programming Library: TSO Extensions Planning and Installation, Volume 1, SC28-1379

System Programming Library: TSO Extensions User Exits and Modifications, Volume 2, SC28-1380

System Programming Library: TSO Extensions Command and Macro Reference, Volume 3, SC28-1381

Assembler H Version 2 Application Programming: Language Reference, GC26-4037

Interactive System Productivity Facility/Program Development Facility for MVS: Reference, SC34-2089

Contents

Basic Information for Using TSO 1
Using a TSO Command 1
Positional Operands 1
Keyword Operands 1
Syntax Notational Conventions 2
Abbreviating Keyword Operands 3
Comments 4
Line Continuation 4
Delimiters 4
Using the HELP Command 5
Using Commands for VSAM and Non-VSAM Data Sets 6
Using Other TSO Commands 6

The Commands and Subcommands 7
ALLOCATE Command 8
ATTRIB Command 32
CALL Command 40
CANCEL Command 42
DELETE Command 44
EDIT Command 48
Subcommands for EDIT 55
Modes of Operation 58
Input Mode 58
Edit Mode 61
Changing from One Mode to Another 63
Data Set Disposition 63
Tabulation Characters 63
Executing User-Written Programs 64
Terminating the EDIT Command 64
Recovering an EDIT Work File 64
Checkpointing a Data Set 64
Recovering After a System Failure 65
Recovering After an Abend 66
Recovering After a Terminal Line Disconnect 67
ALLOCATE Subcommand of EDIT 68
ATTRIB Subcommand of EDIT 69
BOTTOM Subcommand of EDIT 70
CHANGE Subcommand of EDIT 71
Quoted-String Notation 72
Combinations of Operands 73
CKPOINT Subcommand of EDIT 77
COPY Subcommand of EDIT 79
Messages 81

DELETE Subcommand of EDIT 87
DOWN Subcommand of EDIT 89
END Subcommand of EDIT 90
EXEC Subcommand of EDIT 91
FIND Subcommand of EDIT 92
FREE Subcommand of EDIT 94
HELP Subcommand of EDIT 95
INPUT Subcommand of EDIT 96
INSERT Subcommand of EDIT 98
Insert/Replace/Delete Function of EDIT 100
How the System Interprets the Operands 100
LIST Subcommand of EDIT 102
MOVE Subcommand of EDIT 104
Messages 106
PROFILE Subcommand of EDIT 112
RENUM Subcommand of EDIT 113
RUN Subcommand of EDIT 115
SAVE Subcommand of EDIT 118
SCAN Subcommand of EDIT 121
SEND Subcommand of EDIT 123
SUBMIT Subcommand of EDIT 124
TABSET Subcommand of EDIT 128
TOP Subcommand of EDIT 130
UNNUM Subcommand of EDIT 131
UP Subcommand of EDIT 132
VERIFY Subcommand of EDIT 133
END Command 134
EXEC Command 135
FREE Command 140
HELP Command 144
LINK Command 148
LISTALC Command 156
LISTBC Command 159
LISTCAT Command 160
LISTDS Command 164
LOADGO Command 167
LOGOFF Command 172
LOGON Command 173
OUTPUT Command 177
OUTPUT Subcommands 183
CONTINUE Subcommand of OUTPUT 184
END Subcommand of OUTPUT 186
HELP Subcommand of OUTPUT 187
SAVE Subcommand of OUTPUT 188
PROFILE Command 189
PROTECT Command 196
Passwords 199
Types of Access 199
Password Data Set 200
RECEIVE Command 201
RENAME Command 208
RUN Command 210
SEND Command 214
STATUS Command 217

SUBMIT Command 218
TERMINAL Command 222
TEST Command 227
When to Use TEST 230
Addressing Conventions Associated with TEST 231
Restrictions on Use of Symbols 235
External Symbols 235
Internal Symbols 236
Addressing Considerations 236
Examples of Valid Addresses in TEST Subcommands 236
31-Bit Addressing Considerations Associated with TEST (MVS/XA Only) 237
Programming Considerations Associated with TEST When Using the Virtual Fetch Services (MVS/XA Only) 237
Programming Considerations Associated with TEST for Use in a Cross-Memory Environment 238
TEST Subcommands 239
ALLOCATE Subcommand of TEST (MVS/XA Only) 243
AND Subcommand of TEST (MVS/XA Only) 244
Assignment of Values Function of TEST 247
AT Subcommand of TEST 251
ATTRIB Subcommand of TEST (MVS/XA Only) 255
CALL Subcommand of TEST 256
CANCEL Subcommand of TEST (MVS/XA Only) 259
COPY Subcommand of TEST 260
DELETE Subcommand of TEST 263
DROP Subcommand of TEST 264
END Subcommand of TEST 265
EQUATE Subcommand of TEST 266
EXEC Subcommand of TEST (MVS/XA Only) 269
FREEMAIN Subcommand of TEST 270
GETMAIN Subcommand of TEST 272
GO Subcommand of TEST 274
HELP Subcommand of TEST 276
LINK Subcommand of TEST (MVS/XA Only) 277
LIST Subcommand of TEST 278
LISTALC Subcommand of TEST (MVS/XA Only) 283
LISTBC Subcommand of TEST (MVS/XA Only) 284
LISTCAT Subcommand of TEST (MVS/XA Only) 285
LISTDCB Subcommand of TEST 286
LISTDEB Subcommand of TEST 288
LISTDS Subcommand of TEST (MVS/XA Only) 291
LISTMAP Subcommand of TEST 292
LISTPSW Subcommand of TEST 294
LISTTCB Subcommand of TEST 296
LOAD Subcommand of TEST 299
OFF Subcommand of TEST 301
OR Subcommand of TEST (MVS/XA Only) 303
PROFILE Subcommand of TEST (MVS/XA Only) 306
PROTECT Subcommand of TEST (MVS/XA Only) 307
QUALIFY Subcommand of TEST 308
RENAME Subcommand of TEST (MVS/XA Only) 311
RUN Subcommand of TEST 312
SEND Subcommand of TEST (MVS/XA Only) 314

STATUS Subcommand of TEST (MVS/XA Only) 315
SUBMIT Subcommand of TEST (MVS/XA Only) 316
TERMINAL Subcommand of TEST (MVS/XA Only) 317
UNALLOC Subcommand of TEST (MVS/XA Only) 318
WHERE Subcommand of TEST 319
TIME Command 322
TRANSMIT Command 323
Data Encryption Function of TRANSMIT and RECEIVE 324
Logging Function of TRANSMIT and RECEIVE 324
NAMES Data Set Function 325
Control Section Tags 326
Tag Definitions 326
Nicknames Section Tags 328
Tag Definitions 328
TSOEXEC Command 336
WHEN Command 337

The CLIST Statements 339
ATTN CLIST Statement 340
CLOSFILE CLIST Statement 342
CONTROL CLIST Statement 343
DATA-ENDDATA CLIST Sequence 346
DATA PROMPT-ENDDATA Sequence 347
DO-WHILE-END CLIST Sequence 349
ERROR CLIST Statement 350
EXIT CLIST Statement 352
GETFILE CLIST Statement 353
GLOBAL CLIST Statement 354
GOTO CLIST Statement 355
IF-THEN-ELSE CLIST Statement 356
OPENFILE CLIST Statement 357
PROC CLIST Statement 358
PUTFILE CLIST Statement 359
READ CLIST Statement 360
READDVAL CLIST Statement 361
RETURN CLIST Statement 362
SET CLIST Statement 363
TERMIN CLIST Statement 364
WRITE and WRITENR Statements 365

Quick Reference Guide to Commands and Statements 367

Index 369

Figures

1. Commands Preferred for VSAM/Non-VSAM Data Sets 6
2. Allocating and Creating Input Data Sets 40
3. Subcommands of the EDIT Command 56
4. Default Values for LINE or LRECL and BLOCK or BLKSIZE Operands 57
5. Entering Blank Lines Into Your Data Set 60
6. How EDIT Subcommands Affect the Line Pointer Value 62
7. Sample Edit Session Using the CKPOINT Subcommand and the RECOVER Operand of EDIT 66
8. Default Tab Settings 128
9. Information Available Through the HELP Command 147
10. System Defaults for Control Characters 189
11. UPT/PSCB Initialization Table in the Background 194
12. Source Statement/Program Product Relationship 210
13. TSO Commands and Statements and Their Uses 367

Summary of Amendments

Summary of Amendments
for SC28-1307-1
TSO Extensions Release 2

This edition contains several minor technical corrections and service updates throughout the book. Parts of the TRANSMIT and RECEIVE commands have been rewritten for clarity.

Summary of Amendments
for SC28-1307-0
as Updated December 14, 1984
by TNL SN28-1030

This Technical Newsletter contains service updates to the CANCEL, STATUS, and SUBMIT commands.

Basic Information for Using TSO

Using a TSO Command

A command consists of a command name usually 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 to be renamed and specify the new name:

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

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

Positional Operands

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

```
EDIT tfknu47.data
```

where tfknu47.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

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. The value is entered within parentheses following the keyword. For example, a typical keyword operand with a value is:

```
LINESIZE(integer)
```

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

```
LINESIZE(80)
```

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

Syntax Notational Conventions

The following paragraphs describe the notation that this book uses to define the command syntax and format.

1. The set of symbols listed below is used to define the format, but you cannot type them in the actual statement.

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

   The special uses of these symbols are explained in the following paragraphs.

2. You can type uppercase letters, numbers, and the set of symbols listed below in an actual command exactly as shown in the statement definition.

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

3. Lowercase letters and symbols appearing in a command definition represent variables for which you can substitute specific information in the actual command.

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

4. Hyphens join lowercase words and symbols to form a *single* variable.

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

5. The default option is indicated by an underscore. If you do not specify anything, you automatically get the default option. For example,

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

   indicates you can select DISCONNECT or HOLD. However, if no operand is specified, the default is DISCONNECT.

6. Braces group related items, such as alternatives. You must choose one of the items enclosed within the braces. For example,

   ```
   CALL   { dsname                 }
          { dsname (membername)    }
   ```

   indicates if you select dsname (membername), the result is CALL dsname (membername).

7. Brackets also group related items. However, everything within the brackets is optional and can be omitted. For example,

   ```
   PROTECT    data-set-name [ PWREAD   ]
                            [ NOPWREAD ]
   ```

   indicates you can choose one of the items enclosed within the brackets or you can omit both items within the brackets.

8. An ellipsis indicates the preceding item or group of items can be repeated more than once in succession. For example,

   ```
   DELETE (entryname[/ password][...])
   ```

   indicates an entry name and associated optional password you can repeat any number of times in succession.

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)

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

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

or

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

Line Continuation

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

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

or

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

Delimiters

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

Using the HELP Command

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

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

```
help
```

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.

Figure 1 shows recommended commands, by function, for VSAM and non-VSAM data sets. Numbers in parentheses after the commands indicate order of preference. Program product commands are identified with an asterisk (*). Refer to *Access Method Services* for commands not covered in this book.

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

Figure 1. Commands Preferred for VSAM/Non-VSAM Data Sets

Using Other TSO Commands

The Session Manager TSO commands SMCOPY, SMFIND, and SMPUT are not described in this book. For complete descriptions of these commands, see *TSO Extensions Session Manager Program Reference.*

The Commands and Subcommands

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

ALLOCATE Command

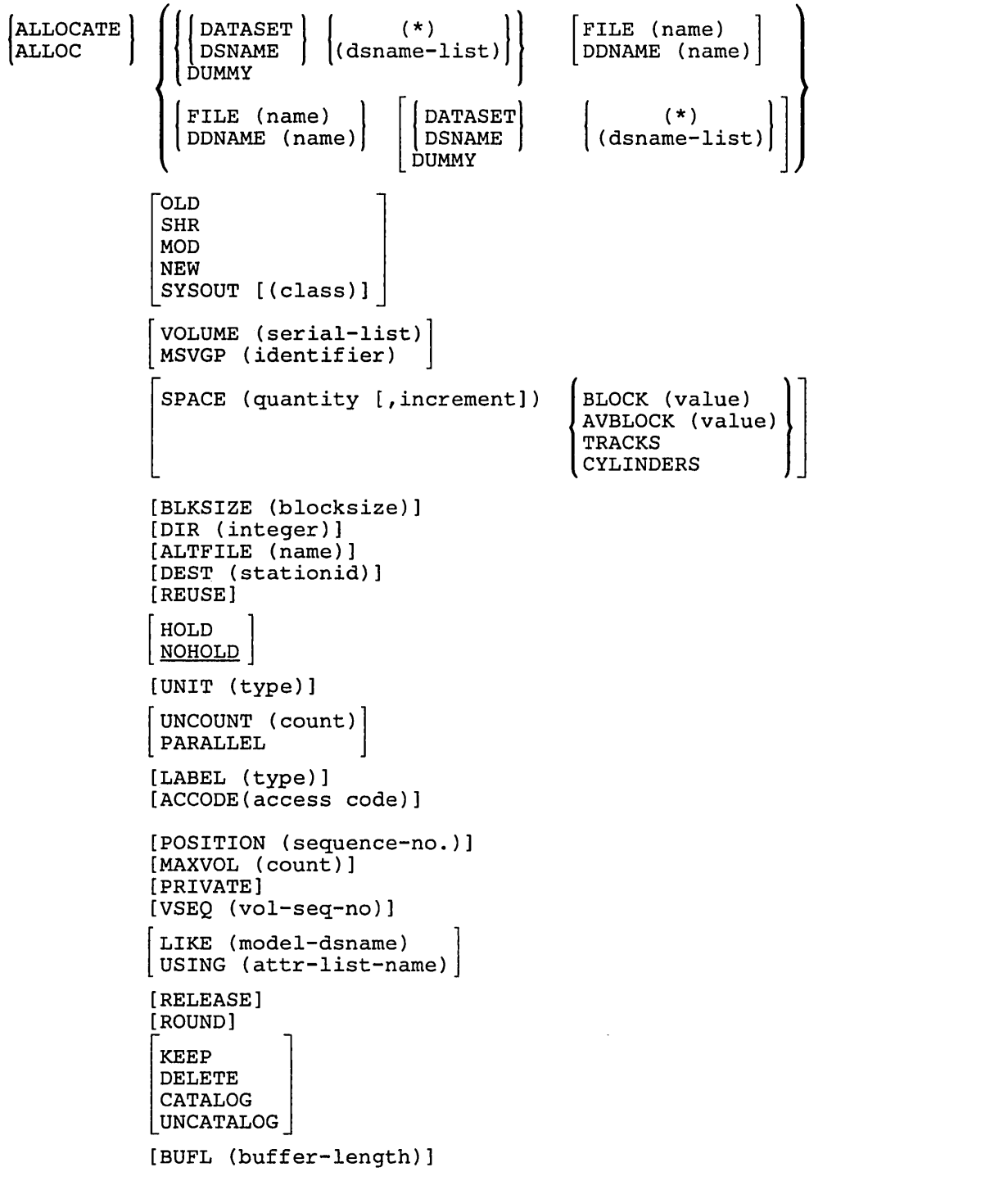
Use the ALLOCATE command or the ALLOCATE subcommand of EDIT (the subcommand's function and syntax are identical to the ALLOCATE command) to dynamically allocate the data sets required by a program that you intend to execute.

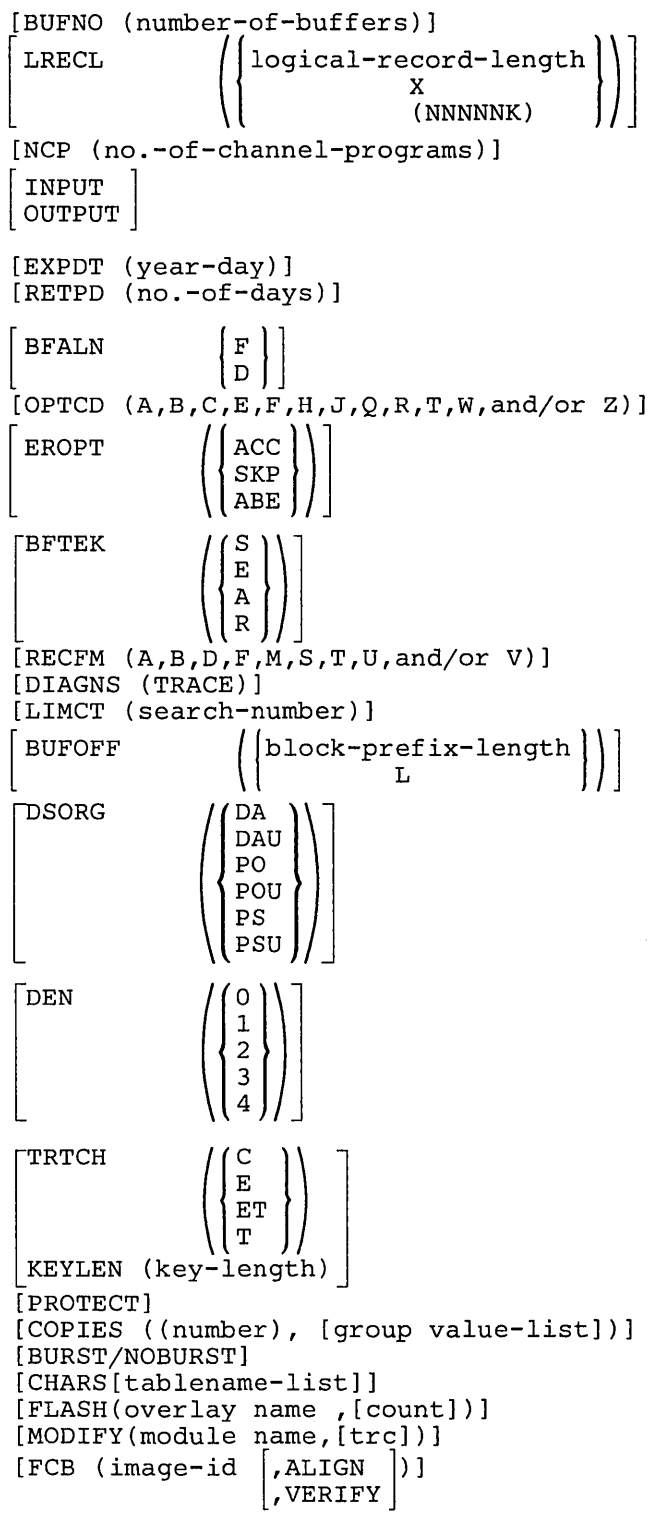
You can specify data set attributes for non-VSAM data sets that you intend to allocate dynamically in several ways:

- 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.


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


[BUFNO (number-of-buffers)]
LRECL
logical-record-length
X
(NNNNNK)
[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,and/or Z)]
EROPT
ACC
SKP
ABE
BFTEK
S
E
A
R
[RECFM (A,B,D,F,M,S,T,U,and/or V)]
[DIAGNS (TRACE)]
[LIMCT (search-number)]
BUFOFF
block-prefix-length
L
DSORG
DA
DAU
PO
POU
PS
PSU
DEN
0
1
2
3
4
TRTCH
C
E
ET
T
KEYLEN (key-length)
[PROTECT]
[COPIES ((number), [group value-list])]
[BURST/NOBURST]
[CHARS[tablename-list]]
[FLASH(overlay name ,[count])]
[MODIFY(module name,[trc])]
[FCB (image-id ,ALIGN ,VERIFY)]

DATASET(dsname-list or *) or DSNAME(dsname-list or *)
specifies the name of the data set that is to be allocated. If a list of data set names is entered, ALLOCATE 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.

1. Data sets residing on the same physical tape volume cannot be allocated concurrently.

2. 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 data sets that you can concatenate is 255. This maximum applies to sequential data sets. For more information on the maximum number of partitioned data sets that you can concatenate, see *Data Administration Guide*. The data sets to be concatenated must all have the same record format (RECFM). The block size (BLKSIZE) of the first data set in the concatenation must be greater than or equal to the block size of each of the other data sets in the same concatenation.

   - The data set group is concatenated. You must free it to deconcatenate it. The file name 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 the following: DATASET/DSNAME, FILE/DDNAME, and status operands.

3. To allocate a member of a generation data group, specify the fully qualified data set name, including the generation number.

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 dataset, only the following operands are processed:

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

FILE(name) or 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 DD name in the data control block (DCB) that is associated with the data set.) For PL/I, this name is the file name in a DECLARE statement and has the form DCL 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 DD names unless you want to make use of the facilities those names represent to the system. See *JCL* for more information on the following special DD names:

```
SYSMDUMP
SYSUDUMP
SYSCHK
SYSCKEOV
SYSABEND
```

See *System Programming Library: Job Management* (MVS/370) or *System Programming Library: System Modifications* (MVS/XA) for the following special DD names:

```
JOBCAT
JOBLIB
STEPCAT
STEPLIB
```

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 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 you want to append data to the end of the data set. If the data set does not exist, a new data set is created 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) indicates the data set does not exist and it is to be created. For new partitioned data sets, you must specify the DIR operand. A NEW data set is kept and cataloged if you specify a data set name. If you do not specify a data set name, it is deleted when you free it or log off.

SYSOUT[(class)]
indicates 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.

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

```
ALLOCATE DDNAME, SYSOUT, DEST, HOLD, NOHOLD, COPIES,
BURST/NOBURST, CHARS, FLASH, MODIFY, FCB
```

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.

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

SPACE(quantity, increment)
specifies the amount of space to be allocated for a new data set. If you omit this operand or the primary space quantity, the system uses the IBM-supplied default value of SPACE(10,50) AVBLOCK (1000). However, your installation might have changed the default. For more information about default space, see *System Programming Library: System Macros and Facilities, Volume 1*.

To indicate the unit of space for allocation, you must specify one of the following: BLOCK(value) or BLKSIZE(value), AVBLOCK(value), TRACKS, or CYLINDERS. The amount of space requested is determined as follows:

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

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

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

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

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

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

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

BLOCK(value)
specifies the average length (in bytes) of the records 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 of (10,50) AVBLOCK(1000) if 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.
- 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.

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

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

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

BLKSIZE(blocksize)
specifies the data control block (DCB) block size for the data set. The maximum allowable decimal value for block size recorded in the DCB is 32,760. The DCB block size is determined in one of the following ways:

- From the attribute list if USING is specified. The BLKSIZE operand on ALLOCATE cannot be used for the DCB block size.
- From the BLKSIZE operand specified on ALLOCATE.
- From the model data set if LIKE is specified and BLKSIZE is not specified on ALLOCATE.
- From the BLOCK operand if neither USING, BLKSIZE, nor LIKE is specified.

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. If you specify:

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

- RECFM(V), then the block size must be equal to or greater than the largest block in the data set. (Note: For unblocked variable-length records, the size of the largest block must allow space for the four-byte block descriptor word in addition to the largest logical record length. The logical record length must allow space for a four-byte record descriptor word.)

- RECFM(V,B), then the block size must be equal to or greater than the largest block in the data set. 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) and BLKSIZE(80), then one character is truncated from the line. That character (the last byte) is reserved for an attribute character.

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.

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

DEST(stationid)
specifies a remote work station to which SYSOUT data sets are directed upon deallocation. The DEST operand is the one to eight character name of the remote work station receiving the SYSOUT data set.

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

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
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 unit type 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 address. 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. This condition exists only if the coded type and class are the same as the cataloged type and class.

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

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

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

ACCODE(access code)
specifies or changes the accessibility code for an ANSI output tape data set. The purpose of the code is to protect the ANSI data set from unauthorized use. Up to eight characters (A-Z) are permitted in the access code, but only the first character is validated by ANSI. The first character must be an upper case alphabetic character. Password protection is supported for ANSI tape data sets under the PASSWORD/NOPWREAD options on the LABEL operand. Password access overrides any ACCODE value if both options are specified.

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

MAXVOL(count)
specifies the maximum number (1-255) of volumes a data set can reside upon. 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. VSEQ should only be specified when the data set is cataloged.

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.

When the ALLOCATE command assigns attributes to a new data set, it copies all of the following attributes from the model data set:

Primary space quantity (SPACE)
Secondary space quantity (SPACE)
Space unit (BLOCK, AVBLOCK, TRACKS, CYLINDERS)
Directory space quantity (DIR)
Data set organization (DSORG)
Record format (RECFM)
Optional services code (OPTCD) - for ISAM data sets only
Logical record length (LRECL)
Key length (KEYLEN)
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 as well as 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:

- The LIKE and USING operands are mutually exclusive.
- 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.

- 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.

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.

Note: The DCB operands (operands that are also on the ATTRIB command) cannot be specified with the USING operand.

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

Note: 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.

Note: A command processor can modify the final disposition of the following operands:

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 after it is freed.

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. If 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 limit established when the operating system was generated. The following table shows the condition that requires you to include this operand.

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

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

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

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

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

LRECL(NNNNNK) allows users of 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 can be 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 32,756 bytes.

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

INPUT
specifies 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.

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. You must specify the year and day in the form yyddd, where yy is a two digit decimal number for the year and ddd is a three digit decimal number for the day of the year using the Julian day format. For example, January 1, 1984 is 84001 and December 31, 1984 is 84366.

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

BFALN $\left(\left\{\begin{matrix}\mathbf{F}\\\mathbf{D}\end{matrix}\right\}\right)$
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/or Z)
specifies the following optional services that you want the system to perform. (See also the OPTCD subparameter of the DCB parameter in *JCL* for a detailed discussion of these services.)

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.

J specifies the character after the carriage control character is the table reference character for that line. The table reference character tells TSO 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 $\left(\left\{\begin{matrix}\textbf{ACC}\\\textbf{SKP}\\\textbf{ABE}\end{matrix}\right\}\right)$

specifies the option 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 $\left(\left\{\begin{matrix}\textbf{S}\\\textbf{E}\\\textbf{A}\\\textbf{R}\end{matrix}\right\}\right)$

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. See also the RECFM subparameter of the DCB parameter in *JCL* for a detailed discussion of the formats and characteristics.

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.

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 $\left(\left\{\begin{matrix}\textbf{block-prefix-length}\\ \textbf{L}\end{matrix}\right\}\right)$
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 $\left(\left\{\begin{matrix}\textbf{DA}\\ \textbf{DAU}\\ \textbf{PO}\\ \textbf{POU}\\ \textbf{PS}\\ \textbf{PSU}\end{matrix}\right\}\right)$
specifies the data set organization as follows:

DA	direct access
DAU	direct access unmovable
PO	partitioned organization
POU	partitioned organization unmovable
PS	physical sequential
PSU	physical sequential unmovable

DEN ((0 | 1 | 2 | 3 | 4))

specifies the magnetic tape density as follows:

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

TRTCH ((C | E | T | ET))

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

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

This operand is mutually exclusive with KEYLEN.

KEYLEN(key-length)

specifies the length in bytes of each of the keys used to locate blocks of records in the data set when the data set resides on a direct access device. The key length must not exceed 255 bytes. If an existing data set has standard labels, you can omit this operand and let the system retrieve the key length from the standard label. If a key length is not supplied by any source before you issue an OPEN macro instruction, a length of zero (no keys) is assumed. This operand is mutually exclusive with TRTCH.

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.
- 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 invalid 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-list])
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 which can be requested is subject to an installation limit. You can specify up to 8 group values. See *JCL* for more information.

- 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 3800 output. SYSOUT is the only valid data set status that can be specified with the BURST operand.

CHARS[tablename-list]
specifies a request for name or names of character arrangement tables (fonts) for printing a data set with the 3800 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 can be specified with the CHARS operand.

FLASH(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. FLASH also allows you to specify the number of copies on which the overlay is to be printed. Copies (0-255) is optional. SYSOUT is the only valid data set status that can be specified with the FLASH operand.

MODIFY(module-name,[trc])
specifies the name of a copy modification module, which is loaded into the 3800 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. TSO defines and stores the module in SYS1.IMAGELIB. The module name can contain 1 to 4 alphameric or national characters.

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

TRC corresponds to the character set(s) specified on CHARS (0-3). 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 can be specified with the MODIFY operand.

FCB(image-id [**,ALIGN**] **)**
[**,VERIFY**]

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 a 3800 printer or a IBM 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 on the forms control buffer, see *System Programming Library: Data Management*, *Programming Support for the IBM 3505 Card Reader and IBM 3525 Card Punch*, or *IBM 3800 Printing Subsystem Programmer's Guide*.

image-id

specifies 1-4 alphameric or national characters 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 a 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 desired one. The VERIFY subparameter is ignored for SYSOUT data sets.

Example 1

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

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

Example 2

Operation: 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

Operation: Allocate an existing data set that is not cataloged.

Known:

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

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

Example 4

Operation: 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

Operation: 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

Operation: 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

Operation: Allocate a new partitioned data set with space allocated in blocks.

Known:

The new data set name: MOSER7.EX2.DATA
The block length: 1000 bytes
The DCB block size: 200
The number of directory blocks: 2
The record format: Fixed Block

```
alloc da(ex2.data) dsorg(po) block(1000) space(10,10) +
lrecl(100) blksize(200) dir(2) recfm(f,b) new
```

Example 8

Operation: 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

Operation: 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

Operation: 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

Operation: 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

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

Known:

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

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

Example 13

Operation: 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 DD name: SYSLIB

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

Example 14

Operation: 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: 2400

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

Example 15

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

Known:

The DD name: OUTPUT
The FCB image desired: STD1
The number of copies: 10

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

Example 16

Operation: 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: 2400

```
alloc da(tape2.data) unit(2400) label(sl) position(1) +
volume(tapev2) protect new
```

Example 17

Operation: 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

Operation: Concatenate a number of data sets.

Known:

The data set names: A.CLIST, B.CLIST, C.CLIST
The DD name: 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.

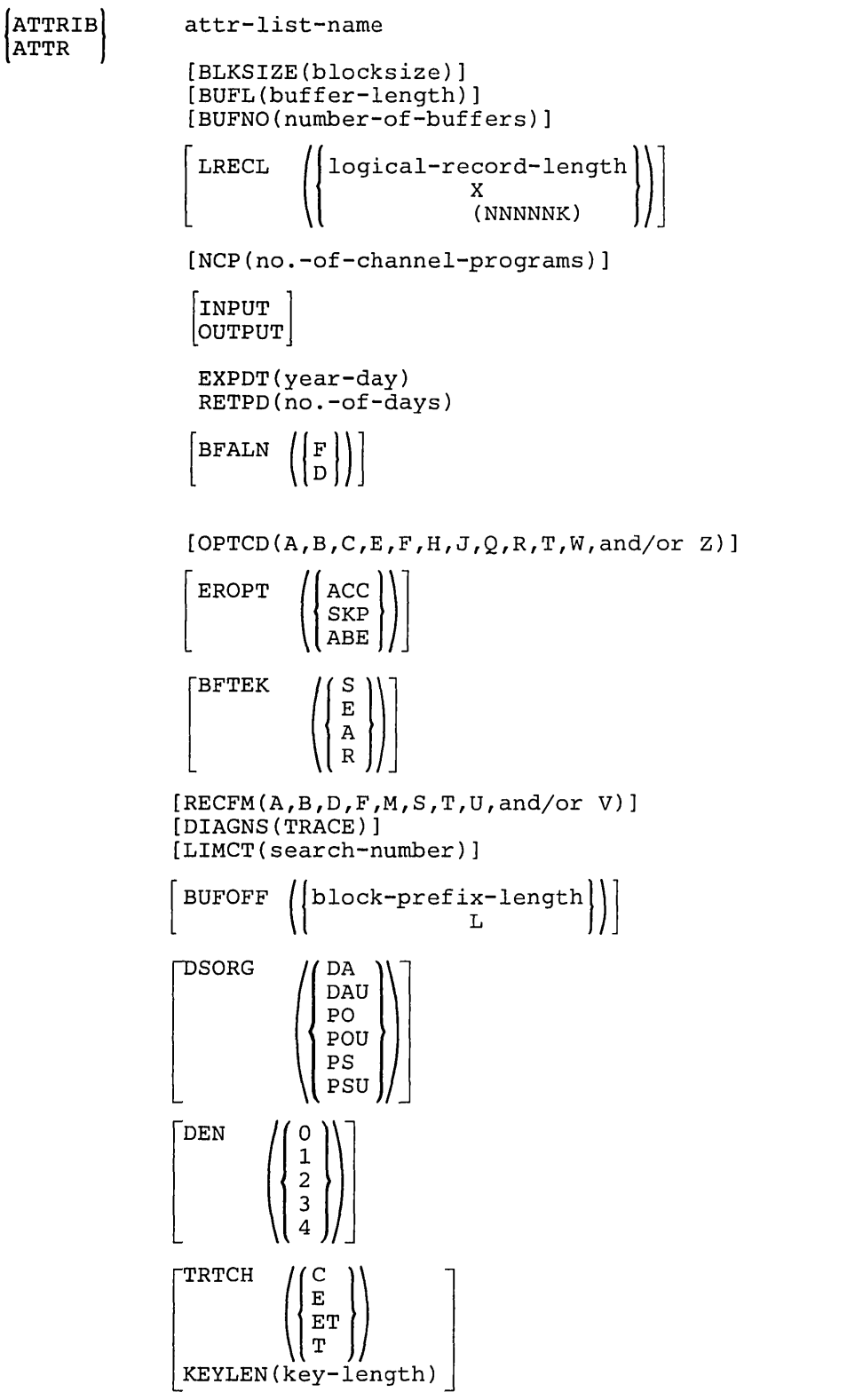
2. Specify the REUSE operand with the ALLOCATE command when you concatenate. The REUSE operand specifies the file name being allocated is to be freed and reallocated if it is currently in use.

ATTRIB Command

Use the ATTRIB command to build a list of attributes for non-VSAM data sets that you intend to allocate dynamically. During the remainder of your terminal session, you can have the system refer to this list for data set attributes when you enter the ALLOCATE command. The ALLOCATE command converts the attributes into DCB operands and LABEL operands for data sets being allocated. See also the subparameters of the DCB parameter in *JCL*.

The ATTRIB command allocates a file with the same name as your attribute-list-name. You can use the LISTALC command with the STATUS 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
ATTR
attr-list-name
[BLKSIZE(blocksize)]
[BUFL(buffer-length)]
[BUFNO(number-of-buffers)]
LRECL
logical-record-length
X
(NNNNNK)
[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,and/or Z)]
EROPT
ACC
SKP
ABE
BFTEK
S
E
A
R
[RECFM(A,B,D,F,M,S,T,U,and/or V)]
[DIAGNS(TRACE)]
[LIMCT(search-number)]
BUFOFF
block-prefix-length
L
DSORG
DA
DAU
PO
POU
PS
PSU
DEN
0
1
2
3
4
TRTCH
C
E
ET
T
KEYLEN(key-length)

attr-list-name
specifies the name for the attribute list. You can specify this name later as an operand of the ALLOCATE command. The name must consist of one through eight alphameric and/or national characters, must begin with an alphabetic or national character, and must be different from all other attribute list names and DD names 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 32,760 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) and BLKSIZE(80), then one character is truncated from the line, that character (the last byte) is reserved for an attribute character.

BUFL(buffer-length)
specifies the length, in bytes, of each buffer in the buffer pool. Specify 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. Specify a decimal number for number-of-buffers. The number must not exceed 255, and you might be limited to a smaller number of buffers

depending on the limit established at system generation time. The following table shows the condition that requires you to include this operand.

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

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

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

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

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

LRECL(NNNNNK) allows users of 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 32,756 bytes.

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

INPUT
specifies 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.

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. You must specify the year and day in the form yyddd, where yy is a two digit decimal number for the year and ddd is a three digit decimal number for the day of the year using Julian day format. For example, January 1, 1984 is 84001 and December 31, 1984 is 84366.

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

BFALN $\left(\left\{\begin{matrix}\mathbf{D}\\\mathbf{F}\end{matrix}\right\}\right)$
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/or Z)
specifies the following optional services that you want the system to perform. See also the OPTCD subparameter of the DCB parameter in *JCL* for a detailed discussion of these services.

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 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 $\left(\left\{\begin{matrix}\textbf{ACC}\\\textbf{SKP}\\\textbf{ABE}\end{matrix}\right\}\right)$

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 $\left(\left\{\begin{matrix}\textbf{S}\\\textbf{E}\\\textbf{A}\\\textbf{R}\end{matrix}\right\}\right)$

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. See also the RECFM subparameter of the DCB parameter in *JCL* for a detailed discussion of the formats and characteristics.

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 can 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).

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

C data conversion with odd parity and no translation

E even parity with no translation and no conversion

T odd parity and no conversion; BCD to EBCDIC translation when reading and EBCDIC to BCD translation when writing

ET even parity and no conversion; BCD to EBCDIC translation when reading and EBCDIC to BCD translation when writing

This operand is mutually exclusive with KEYLEN.

KEYLEN(key-length)

specifies the length, in bytes, of each of the keys used to locate blocks of records in the data set when the data set resides on a direct access device. The key length must not exceed 255 bytes. If an existing data set has standard labels, you can omit this operand and let the system retrieve the key length from the standard label. If a key length is not supplied by any source before you issue an OPEN macro instruction, a length of zero (no keys) is assumed. This operand is mutually exclusive with TRTCH.

Example 1

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

Known:

The following attributes correspond to the DCB 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 32,756 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) 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) to be processed. It must be a member of a partitioned data set.

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 one contains the address of a fullword. The three low order bytes of this fullword contain the address of a halfword field. This halfword field is the count of the number of bytes of information contained in the parameter list. The parameters immediately follow the halfword field.

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

CALL 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 as well as the data itself). Once the data set is created, you can use the CALL command to execute the program that accesses the SYSIN data. Figure 2 illustrates the allocation and creation of input data sets:

```
//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 2. Allocating and Creating Input Data Sets

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              {dsname             }
                  {dsname(membername) }

                   ['parameter-string']
```

dsname
specifies the name of a sequential data set to be executed.

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

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

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

```
'wrrid.myprogs.loadmod(a)'
'sys1.linklib(ieuasm)'
```

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

Example 1

Operation: Execute a load module.

Known:

The name of the load module: JUDAL.PEARL.LOAD(TEMPNAME)

Parameters: 10,18,23

```
call pearl(tempname) '10,18,23'
```

Example 2

Operation: Execute a load module.

Known:

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

```
call mylib(cos1)
```

Example 3

Operation: Execute a load module.

Known:

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

```
call (sin1)
```

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 in conjunction 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     (jobname[(jobid)]-list)
                [ NOPURGE ]
                [ PURGE   ]
```

(jobname[(jobid)]-list)
specifies the names of the jobs that you want to cancel. The job names must consist of your user identification plus one or more alphameric characters up to a maximum of eight characters unless the IBM-supplied exit has been replaced by your installation.

The optional job ID subfield may consist of one to eight alphameric characters (the first character must be alphabetic or national). The job ID is a unique job identifier assigned by the job entry subsystem (JES) at the time the job was submitted to the system. The job ID is needed if you have submitted two jobs with the same name.

Note the following:

- 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.
- Jobs controlled by the subsystems are considered started tasks and cannot be cancelled by the CANCEL command.

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

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

Example 1

Operation: Cancel a batch job.

Known:

The name of the job: JE024A1

```
cancel je024a1
```

Example 2

Operation: Cancel several batch jobs.

Known:

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

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

DELETE Command

Use the DELETE command to delete one or more data set entries or one or more members of a partitioned data set. The catalog entry for a partitioned data set is removed only when the entire partitioned data set is deleted. The system deletes a member of a partitioned data set by removing the member name from the directory of the partitioned data set.

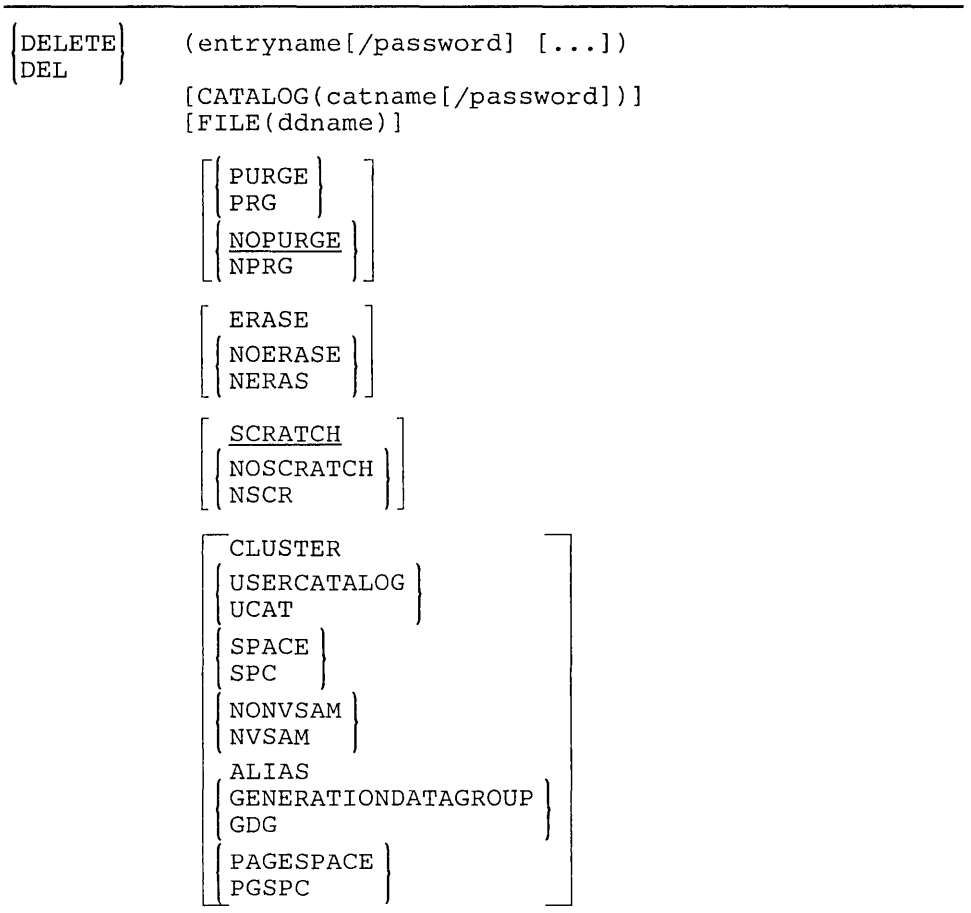
Members of a partitioned data set and aliases for any members must each be deleted explicitly. That is, when you delete a member, the system does not remove any alias names of the member. Likewise, when you delete an alias name, the member itself is not deleted.

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

For MVS, the original TSO DELETE command has been replaced by the Access Method Services command with the same name. Note that when you delete a data set, you must also free the allocated DD names. If you want to modify VSAM objects or use the other Access Method Services from a terminal, see *Access Method Services*. For error message information, see *Message Library: System Messages*.

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

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



entryname[/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:

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

If you specify

```
delete source2.*
```

the only data set entry remaining is:

```
VACOT.SOURCE.PLI
```

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

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

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

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

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

NOPURGE or 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. If neither PURGE nor NOPURGE is coded, NOPURGE is the default.

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

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 if neither SCRATCH nor NOSCRATCH is specified.

NOSCRATCH or 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 or 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
specifies the entry to be deleted is a 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.

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

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

GENERATIONDATAGROUP or 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 or 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.

Example

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

Known:

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. The data sets might contain:

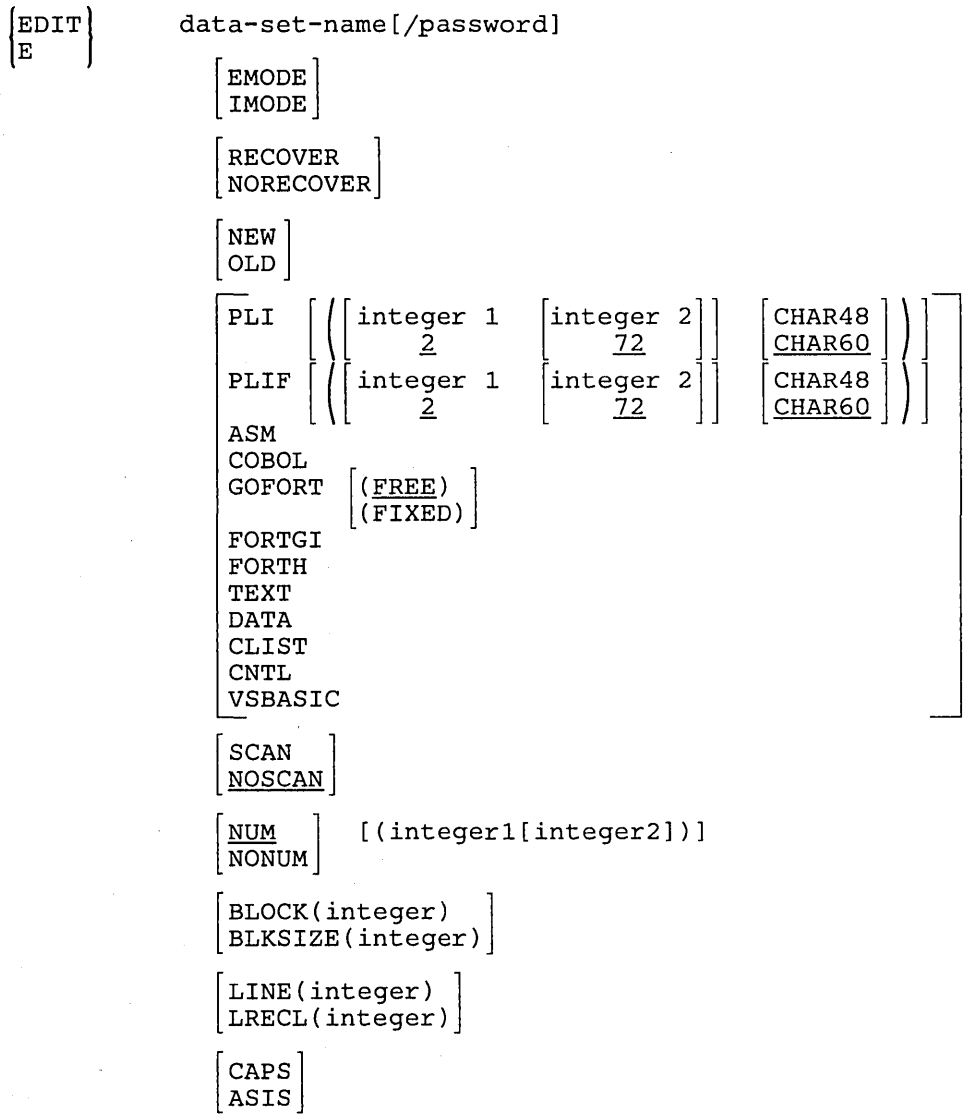
- Source programs composed of program language statements such as PL/I, COBOL, FORTRAN, etc.
- Data used as input to a program.
- Text used for information storage and retrieval.
- Commands, subcommands, CLIST statements and/or data.
- Job control language (JCL) statements for background jobs.

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.

After you edit 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.



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

password
specifies the password associated with the data-set-name. If the password is omitted and the data set is password protected, you 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
specifies the initial mode of entry is edit mode. This is the default for OLD data sets. See "Edit Mode" for more information on this operand.

IMODE
specifies the initial mode of entry is input mode. This is the default for NEW or empty data sets. See "Input Mode" for more information on this operand.

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 "Recovering an EDIT Work File" for more information on this operand.

NORECOVER
specifies that you do not want to recover a work file, even if a recoverable work file exists.

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.

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 (specified at system generation) is also a valid data set type operand and can have subfield parameters defined by your installation (see Figure 4, 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.

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

CHAR48 or 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 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.

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 or FIXED)
specifies the data set identified by the first operand is for statements that are suitable for processing by the Code and Go FORTRAN program product. You can use FORT as an abbreviation for this operand. This is the default value if no other FORTRAN language level is specified with the FORTGI or FORTH operand. FREE specifies the statements are of variable-lengths and do not conform to set column requirements. This is the default value if neither FREE nor FIXED is specified. FIXED specifies 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 quotes).
- The data-set-name operand is a member name only (that is, it is enclosed in parentheses).
- The data-set-name operand does not include a descriptive qualifier or the descriptive qualifier is such that EDIT cannot determine the data set type.

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
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.

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

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

NUM(integer1 integer2)

specifies lines of the data set records are numbered. You can specify integer1 and integer2 for ASM type data sets only. Integer1 specifies, in decimal, the starting column (73-80) of the line number. Integer2 specifies, in decimal, the length (8 or less) of the line number. Integer1 plus integer2 cannot exceed 81. If integer1 and integer2 are not specified, the line numbers assume appropriate default values.

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. The default is NUM.

BLOCK(integer) or BLKSIZE(integer)

specifies the maximum length, in bytes, for blocks of records of a new data set. Specify this operand only when creating a new data set or editing an empty old data set. You cannot change the block size of an existing data set except if the data set is empty. If you omit this operand, it defaults according to the type of data set being created. Default block sizes are described in Figure 4. If different defaults are established at system generation time, the values in Figure 4 might not be applicable. The block size (BLOCK or BLKSIZE), for data sets that contain fixed-length records must be a multiple of the record length (LINE or LRECL). For variable-length records, the block size must be a multiple of the record length plus 4.

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

LINE(integer) or LRECL(integer)

specifies the length of the records to be created for a new data set. Specify this operand only when creating a new data set or editing an empty old data set. The new data set 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 line size defaults according to the type of data set being created. Default line sizes for each data set type can be found in Figure 4. Use this operand in conjunction with the BLOCK or BLKSIZE operand.

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.

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 upper case.
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
```

Subcommands for EDIT

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 in a manner similar to that for a command. Figure 3 contains a summary of each subcommand's function.

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

ALLOCATE	Allocates data sets and 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.
CKPT	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.
FIND	Locates a character string.
FORMAT (available as an optional program product)	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 (available as an optional program product)	Combines all or parts of data sets.
MOVE	Moves records within a data set.
PROFILE	Specifies characteristics of your user profile.
RENUM	Numbers or renumbers lines of data.
RUN	Causes compilation and execution of data set.
SAVE	Retains the data set.
SCAN	Controls syntax checking.
SEND	Allows you to communicate with the system operator and with other terminal users.
SUBMIT	Submits a job for execution in the background.
TABSET	Sets the tabs.
TOP	Sets the pointer to zero value.
UNNUM	Removes line numbers from records.
UP	Moves the pointer toward the start of data set.
VERIFY	Causes current line to be listed whenever the current line pointer changes or the text of the current line is modified.

Figure 3. Subcommands of the EDIT Command

Data Set Type	DSORG	LRECL LINE(n) Default	LRECL LINE(n) Specif.	Block Size BLOCK(n) Default	Block Size BLOCK(n) Specif. (Note 1)	Line Numbers NUM(n,m) Default(n,m) Spec.	CAPS/ASIS Default	CAPS/ASIS CAPS Required
ASM	PS/PO	80	=80	3120	<=default	Last 8 73<=n<=80	CAPS	Yes
CLIST	PS/PO	255	(Note 2)	3120	<=default	(Note 3)	CAPS	Yes
CNTL	PS/PO	80	=80	3120	<=default	Last 8	CAPS	Yes
COBOL	PS/PO	80	=80	400	<=default	First 6	CAPS	Yes
DATA	PS/PO	80	<=255	3120	<=default	Last 8	CAPS	No
FORTGI	PS/PO	255	=80	400	<=default	Last 8	CAPS	Yes
FORTH	PS/PO	255	=80	400	<=default	Last 8	CAPS	Yes
GOFORT	PS/PO	255		3120	<=default	First 8	CAPS	Yes
(Or user supplied data set type - see Note 4)								
PLI	PS/PO	104	‰100	400	<=default	(Note 3)	CAPS	No
PLIF	PS/PO	80	‰100	400	<=default	Last 8	CAPS	Yes
TEXT	PS/PO	255	(Note 2)	3120	<=default	(Note 3)	ASIS	No
VSBASIC	PS/PO	255	=80	3120	<=32,760	First 5	CAPS	Yes

Notes:

1. You can specify the default or maximum allowable block size at system generation time.
2. Specifying a LINE value results in fixed length records with a LRECL equal to the specified value. The specified value must always be equal to or less than the default. If the LINE 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 define the user-defined data set types along with any of the data sets shown above at system generation time by using the EDIT macro. The EDIT macro causes a table of constants to be built, which describes the data set attributes. For more information on how to specify the EDIT macro at system generation time, see *SPL: System Generation Reference.*

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 formats:	Fixed or variable
Logical Record Size:	Less than or equal to 255 characters
Block Sizes:	User specified--must be less than or equal to track length
Sequence Numbers:	V type: First 8 characters F type: Last 8 characters

Figure 4. Default Values for LINE or LRECL and BLOCK or BLKSIZE Operands

Modes of Operation

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

You must specify a data set name when you enter the EDIT command. If you specify the NEW operand, the system places you in the input mode. If you do not specify the NEW operand, you are placed in the edit mode if your specified data set is not empty. If the data set is empty, you are placed in input mode.

If you have limited access to your data set, by assigning a password, you can enter a slash (/) followed by the password of your choice after the data set name operand of the EDIT command.

Entering either EMODE or IMODE operands on the EDIT command overrides the normal mode setting described above. The specification of the RECOVER operand on the EDIT command places you in edit mode upon recovery. Refer to "Recovering an EDIT Work File" for more information about the RECOVER operand.

Input Mode

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

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

Line Numbers: Unless you specify otherwise, the system assigns a line number to each line as it is entered. The default is an interval of 10. Line numbers make editing much easier, because you can refer to each line by its own number.

Each line number consists of up to eight digits, with the significant digits justified on the right and preceded by zeros. Line numbers are placed at the beginning of variable-length records and at the end of fixed-length records. (Exception: Line numbers for COBOL fixed-length records are placed in the first six positions at the beginning of the record.) When you are working with a data set that has line numbers, you can have the new line number listed at the start of each new input line. If you are creating a data set without line numbers, you can request that a prompting character be displayed at the terminal before each line is entered. Otherwise, none is issued.

Input records are converted to uppercase characters, unless you specify the ASIS or TEXT operand. The TEXT operand also specifies that character-deleting indicators and tabulation characters are recognized, but all other characters are added to the data set unchanged.

All Assembler source data sets must consist of fixed-length records, 80 characters in length. These records might not be line numbers. If the records are

line-numbered, the number can be located anywhere within columns 73 to 80 of the stored record (the printed line number always appears at the left margin).

You can create FORTRAN data sets FORTGI and FORTH.

Syntax Checking: You can have each line of input checked for proper syntax. The system checks the syntax of statements for data sets having FORT descriptive qualifiers. Input lines are collected within the system until a complete statement is available for checking.

When an error is found during syntax checking, an appropriate error message is issued and edit mode is entered. You can then take corrective action, using the subcommands. When you want to resume input operations, press the ENTER key. Input mode is then entered and you can continue where you left off. Whenever statements are being checked for syntax during input mode, the system prompts you for each line to be entered unless you specify the NOPROMPT operand for the INPUT subcommand.

Continuation of a Line in Input Mode: In input mode, there are two independent situations that require you to indicate the continuation of a line by ending it with a hyphen or plus sign (that is, a hyphen or plus sign followed immediately by pressing the ENTER key). The situations are:

- The syntax checking facility is being used.
- The data set type is CLIST (variable-length records).

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

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

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

Note that these rules apply only when entering data in input mode. When you use a subcommand (for example, CHANGE or INSERT) to enter data, a hyphen at the end of the line indicates subcommand continuation. The system appends the continuation data to the subcommand.

INPUT Mode in the Background: When the EDIT command is executed in the background and input mode is requested, blank lines should not be entered into the data set. EDIT interprets a blank line as a null line causing a switch from input mode to edit mode. When it is necessary to incorporate blank lines into your data set, certain methods can be followed. One method is to insert an unused character string wherever a blank line is required. Before ending the edit session, insert the CHANGE subcommand to change the character string to blanks. Figure 5 illustrates how this is done:

```
edit   examp4.cntl  new
INPUT
00010 logon user4 proc(proca)
00020 profile prefix(user ID)
00030 edit p data new
00040 line one
00050 @@@@@
00060 line two
00070 @@@@@
00080 line three
00090 @@@@@
00100 line four
00110
00120 c 10 999 /@@@@@// all
00130 list
00140 end save
00150 (null line)
end save
READY
submit examp4.cntl notify jobchar(a)
JOB USER4A(JOB00001) SUBMITTED
READY
```

Figure 5. Entering Blank Lines Into Your Data Set

An alternate method is to specify the operand EMODE on the EDIT command that is to be executed in the background. With this method, each new line of data should be preceded by a line number wherever line number editing is allowed.

To insert a line of data ending in a hyphen in situations where the system would remove the hyphen (that is, while in subcommand mode or in input mode for other than a CLIST data set), enter a hyphen in the next-to-last column, a blank in the last column, and immediately press the ENTER key.

Creating a Data Set: When creating a data set, you must first request input mode. You can do this by entering one of the following:

- The NEW operand on the EDIT command.
- The IMODE operand on the EDIT command.
- The INPUT subcommand while in edit mode.
- The INSERT subcommand with no operands, while in edit mode.
- A null line, if the system is in edit mode.

After you enter the EDIT command with either the NEW or IMODE operands, the system sends you the following message:

```
INPUT
```

For example:

Operation: Add data to an existing data set using the IMODE operand.

Known:

To add data, you want to go into input mode immediately.

Enter:

```
edit cmdproc clist imode
```

Edit Mode

You can enter subcommands to edit data sets when you are in edit mode. You can edit data sets that have line numbers by referring to the number of the line that you want to edit. This is called *line-number editing*. You can also edit data by referring to specific items of text within the lines. This is called *context editing*. A data set having no line numbers can be edited only by context. Context editing is performed by using subcommands that refer to the current line value or a character combination, such as with the FIND or CHANGE subcommands. There is a pointer within the system that points to a line within the data set. Normally, this pointer points to the last line that you referred to. You can use subcommands to change the pointer so that it points to any line of data that you choose. You can then refer to the line that it points to by specifying an asterisk (*) instead of a line number. Figure 6 shows where the pointer points at completion of each subcommand.

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

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

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

Figure 6. How EDIT Subcommands Affect the Line Pointer Value

Changing from One Mode to Another

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

You change from edit mode to input mode when:

- You enter the INPUT subcommand.
- You press the ENTER key before typing anything.
- You enter the INSERT subcommand with no operands.

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

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

You switch from input mode to edit mode when:

- You press the ENTER key before typing anything.
- You cause an attention interruption.
- There is no more space for records to be inserted into the data set and re-sequencing is not allowed.
- An error is discovered by the syntax checker.

Data Set Disposition

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

Tabulation Characters

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

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

Executing User-Written Programs

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

Terminating the EDIT Command

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

Recovering an EDIT Work File

In the event of an abnormal termination, the recovery facility of EDIT enables you to recover changes or modifications made during an edit session (applicable in foreground only). To recover the work file after an abnormal termination has occurred during an edit session, the EDIT command should be reissued specifying the RECOVER operand, along with any any other operands specified initially. This facility is optional to both the installation and/or the TSO user.

Certain specifications must be met before a work file becomes recoverable. They are:

- The installation must not have specified the NORECOVER attribute to your user ID. If the NORECOVER attribute was assigned, the data set is not recoverable.
- To be recoverable, you must enter the PROFILE command containing the RECOVER operand prior to the edit session.

If the conditions above are met, EDIT creates a work file and updates it while your edit session progresses. If the edit session terminates normally, the work file is deleted immediately upon termination. If the edit session is terminated abnormally, the work file is kept and made available at the beginning of your next edit session.

Checkpointing a Data Set

The CKPOINT subcommand of EDIT gives you the ability to automatically checkpoint a data set during the input or modification phase of the edit session. The invocation of checkpointing is controlled through the use of the CKPOINT subcommand. See the CKPOINT subcommand of EDIT for the syntax description.

Recovering After a System Failure

To recover data from your last edit session, issue the EDIT command entering the same data set name that you were working on at the time of the failure and include the RECOVER operand. You are placed in edit mode and the work file data set is used as input for the current edit session. The current line pointer is positioned at the top of the data set.

Note the following:

1. If you specify IMODE upon re-entering your edit session, or if you give a data set disposition of NEW, the recovery feature always puts your session in edit mode.

2. If the RECOVER operand is not specified, you are prompted and given a choice of RECOVER or NORECOVER.

3. If the RECOVER operand is specified and the work file data set name does not match the edited data set name, an error message is issued. You are prompted and given a choice of recovering or not recovering the data set.

4. If the RECOVER operand is specified and the work file data set does not exist, an error message is issued.

The example shown in Figure 7 illustrates the different stages of an edit session and the actions necessary to recover it.

```
READY
profile recover
READY
edit lions old data
EDIT
ckpoint 5
list
00010 THE
00020 EDIT
00030 LOST,
00040 REENTER
00050 COMMAND
00060 AND
00070 SAVE
00080 ENTRY
c 30 /lost,/recovery/
c 40 /reenter/feature/
c 50 /command/saves/
c 60 /and/you/
c 70 /save/time and/
     (System automatically takes a checkpoint after
     fifth line of modifications.)
c 80 /entry/repetition/
     (Assuming system failure has occurred here, your edit
     session will terminate abnormally.  When the system
     is restored, issue the LOGON command and reenter the
     EDIT command including the RECOVER operand.)

edit lions old data recover
EDIT
list
00010 THE
00020 EDIT
00030 RECOVERY
00040 FEATURE
00050 SAVES
00060 YOU
00070 TIME AND
00080 ENTRY
c 80 /entry/repetition/
     (Note:  The last line was not kept.  All other changes
     were kept in the EDIT work file (utility data set)
     making it necessary to reenter only one line.)
```

Figure 7. Sample Edit Session Using the CKPOINT Subcommand and the RECOVER Operand of EDIT

Recovering After an Abend

When an abend occurs after issuing the SAVE subcommand of EDIT because there is not enough space (B37, D37, E37) in your data set or on the volume in which your data set resides, message IKJ52432A is issued. Termination does not occur, even if all attempts to save the data set are unsuccessful. You can respond to the system prompt with one of the following options:

- Enter the SAVE subcommand specifying a different data set name.

- Enter RETAIN to terminate your edit session. The EDIT work file (utility data set) is checkpointed and retained. Recovery is possible at the beginning of your next edit session.

- Enter END to terminate your edit session. With this option, the EDIT work file is not available for recovery at your next edit session.

- Entering any other valid subcommand of EDIT at this time causes the abend to be disregarded and your edit session continues.

Using the RETAIN option allows you to end your edit session and then perform any space recovery measures necessary to obtain additional space. The RECOVER operand on the EDIT command can be used to recover your data set during your next edit session. Refer to "Recovery After System Failure" for the correct procedure.

When your edit session is terminated by a system, operator, or time allocation (abend code X22), the EDIT work file is checkpointed and retained if any modifications were made. This allows you to invoke EDIT's recovery feature after your next logon is issued. For any other abends, you are prompted for END or SAVE through message IKJ52563A. If you do not enter SAVE or END, you are terminated immediately. The EDIT work file is retained if modifications have been made. If SAVE is issued and the attempt is unsuccessful, the edit session is terminated. However, the work file data set is retained if modifications were made, and message IKJ52428I is issued.

See *TSO/E Terminal Messages* for more information about the messages in this section.

Recovering After a Terminal Line Disconnect

If your user profile contains the RECOVER attribute and you are using permanent EDIT work files, EDIT creates a work file during your edit session, which can be used as input to recover any modification made to your data set in the event of a line disconnect or system failure.

Through the use of the CKPOINT subcommand of EDIT and the RECOVER operand of the EDIT command, you are given the opportunity to recover the modifications made to your data set prior to the disconnect.

If your user profile contains the NORECOVER attribute and you are using temporary EDIT work files, the system attempts to copy your edited data set (with all changes) into a data set with an intermediate qualifier name of EDITSAVE. This data set can be edited the next time you log on.

ALLOCATE Subcommand of EDIT

Use the ALLOCATE subcommand to dynamically allocate the data sets required by a program that you intend to execute. Refer to the ALLOCATE command for the description of the syntax and function of the ALLOCATE subcommand.

ATTRIB Subcommand of EDIT

The ATTRIB subcommand of EDIT performs the same function as the ATTRIB command without leaving edit mode. Refer to the ATTRIB command for a description of the syntax and function of the ATTRIB subcommand.

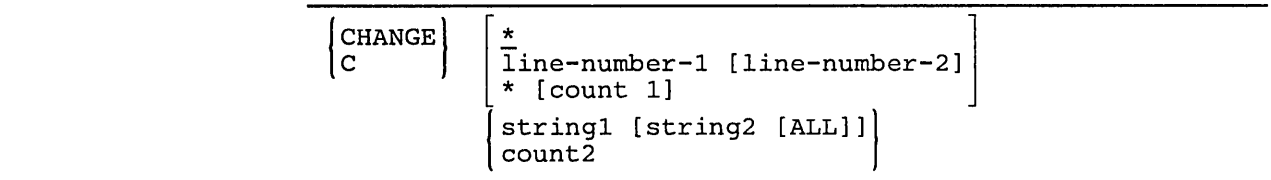
BOTTOM Subcommand of EDIT

Use the BOTTOM subcommand to change the current line pointer to the last line of the data set being edited or to contain a zero value (if the data set is empty). This subcommand can be useful when subsequent subcommands such as INPUT or MERGE must be at the end of the data set.

```
{BOTTOM}
{B     }
```

CHANGE Subcommand of EDIT

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



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

line-number-2
specifies the last line of a range of lines that you want to change. The specified lines are scanned for first occurrence of the sequence of characters specified for string1.

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

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

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

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

string2

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

ALL

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

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, string2 must be followed by the delimiter before typing the ALL operand.

count2

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

Quoted-String Notation

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

- Do not use both special-delimiter and quoted-string notation in the same subcommand.
- Enclose each string with single quotes; for example, 'This is string1' 'This is string2.' Quoted strings must be separated with a blank.
- Use two single quotes to represent a single quote within a character string; for example, 'pilgrim''s progress'.
- Use two single quotes 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'
Delete	!ab!!or!ab!	'ab' ''
Print up to	!ab	'ab'
Place in front of	!!cde!	'' 'cde'

* - 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 count2 operand.

- When you enter two numbers and no other operands, the system assumes that they are line-number-1 and count2 respectively.

- When you enter two operands and the first is a number and the second begins with a character that is not a number, the system assumes that they are line-number-1 and string1.

- When you enter three operands and they are all numbers, the system assumes that they are line-number-1, line-number-2, and count2.

- When you enter three operands and the first two are numbers, but the last begins with a character that is not a number, the system assumes that they are line-number-1, line-number-2, and string1.

Example 1

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

Known:

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

```
change 57 XparameterXoperand
```

Example 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 string1 and string2 examples (i.e. and e.g.) are treated as characters.

Example 3

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

Known:

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

```
change 143 18
```

This form of the subcommand causes the first 18 characters of line number 143 to be displayed at your terminal. You complete the line by typing the new information and enter the line by pressing the ENTER key. All of your changes 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
```

CKPOINT Subcommand of EDIT

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.

```
{CKPOINT}   [value]
{CKP    }
```

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.

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 newvalue
 or
CKP newvalue
```

Example 4

To terminate interval checkpoint, issue the CKPOINT subcommand with a zero value. The entry is:

```
CKPOINT 0
 or
CKP 0
```

COPY Subcommand of EDIT

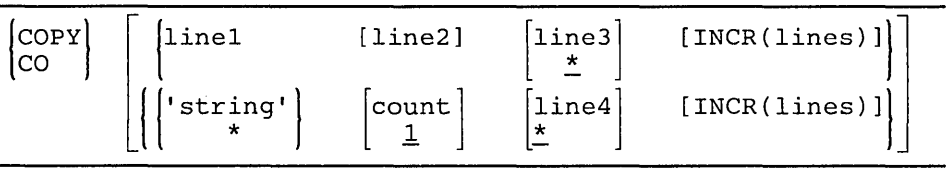
Use the COPY subcommand of EDIT to copy one or more records that exist in the data set being edited. The copy operation moves data from a source location to a target location within the same data set and leaves the source data intact. Existing lines in the target area are shifted toward the end of the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

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

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

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.



line1

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

line2

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

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

line3

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

The value of line3 should not fall in the range from line1 to line2. The target line must not be in the range being copied. Exception: Line3 can be equal to line2.

*****

represents the value of the current line pointer.

INCR(lines)

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

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

'string'

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

count

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

line4

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

Messages

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

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

1. Entering

   ```
   copy * * *
   ```

   causes:

   ```
   INVALID OPERANDS * INVALID FOR COUNT OR END OF RANGE
   SPECIFICATION
   ```

2. Entering

   ```
   copy 10000 *
   ```

   causes:

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

3. Entering

   ```
   copy 'xyz' *
   ```

   causes:

   ```
   INVALID OPERANDS QUOTED STRING NOT FOUND
   ```

4. Entering

   ```
   copy 20 10 *
   ```

   causes:

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

5. Entering

```
copy 20 '*' 100
```

causes:

```
INVALID OPERANDS STRING INVALID FOR END OF RANGE
SPECIFICATION
```

6. Entering

```
copy * 0 100
```

causes:

```
INVALID OPERANDS 0 INVALID FOR COUNT
```

7. Entering

```
copy 10 40 20
```

causes:

```
INVALID OPERANDS TRYING TO MOVE/COPY INTO LINE RANGE
```

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

Example 1

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

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

```
Before:              Enter:                  After:

0010    A            copy     50  50  50     0010    A
0020    BB                                   0020    BB
0030    CCC          or                      0030    CCC
0040    DDDD                                 0040    DDDD
0050    EEEEE        copy     50  50         0050    EEEEE
0060    FFFFFF                        CLP    0060    EEEEE
0070    GGGGGGG      or                      0061    FFFFFF
0080    HHHHHHHH                             0070    GGGGGGG
0090    IIIIIIIII    copy     50             0080    HHHHHHHH
0100    JJJJJJJJJ                            0090    IIIIIIIII
0110    KKKKKKKKKK   or                      0100    JJJJJJJJJ
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
                                            0120    LLLLLLLLLLL
```

Example 4

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

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

```
Before:          Enter:                    After:

NOW IS           top                       NOW IS
THE TIME         copy    'men'  99999999   THE TIME
FOR ALL                                    FOR ALL
GOOD MEN                                   GOOD MEN
TO COME                                    TO COME
TO THE                                     TO THE
AID OF                                     AID OF
THEIR                                      THEIR
COUNTRY                                    COUNTRY
                                    CLP    GOOD MEN
```

Example 5

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

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

```
Before:            Enter:                      After:

0010  A            copy   10  30  100 incr(5)  0010   A
0020  BB                                        0020   BB
0030  CCC          or                          0030   CCC
0040  DDDD                                     0040   DDDD
0050  EEEEE        copy    9  31  100 incr(5)  0050   EEEEE
0060  FFFFFF                                   0060   FFFFFF
0070  GGGGGGG      or                          0070   GGGGGGG
0080  HHHHHHHH                                 0080   HHHHHHHH
0090  IIIIIIIII    copy    1  39  100 incr(5)  0090   IIIIIIIII
0100  JJJJJJJJJJ                              0100   JJJJJJJJJJ
0110  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   JJJJJJJJJ                             0091   FFFFFF
0100   KKKKKKKKKK                            0092   GGGGGGG
0120   LLLLLLLLLLL                           0093   HHHHHHHH
                                             0094   IIIIIIIII
                                             0100   JJJJJJJJJ
                                             0110   KKKKKKKKKK
                                             0120   LLLLLLLLLLL
```

Example 7

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

Known: Data set contains lines 10 through 120; source range is line 70 through line 110; target location is line 50; increment 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   JJJJJJJJ
0100  JJJJJJJJJ                            CLP     0100   KKKKKKKKK
0110  KKKKKKKKKK                                   0101   FFFFFF
0120  LLLLLLLLLLL                                  0102   GGGGGGG
                                                   0103   HHHHHHHH
                                                   0104   IIIIIIIII
                                                   0105   JJJJJJJJJ
                                                   0110   KKKKKKKKK
                                                   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    IIIIIIIII
0030  CCC                                      CLP   0100    JJJJJJJJJJ
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  JJJJJJJJJJ                                     0107    GGGGGGG
0110  KKKKKKKKKK                                     0108    HHHHHHHH
0120  LLLLLLLLLLL                                    0109    IIIIIIIII
                                                     0110    JJJJJJJJJJ
                                                     0111    KKKKKKKKKK
                                                     0120    LLLLLLLLLLL
```

Example 9

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

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

```
Before:              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
```

DELETE Subcommand of EDIT

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

Upon completion of the delete operation, the current line pointer 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.

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

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

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

*
specifies 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.

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 zeroes are not required.

Example 3

Operation: Delete several consecutive lines from the data set.

Known:

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

```
delete 18 36
```

Example 4

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

Known:

The number of lines to be deleted: 18

```
delete * 18
```

Example 5

Operation: Delete all the lines in a data set.

Known:

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

```
top
delete * 100
```

DOWN Subcommand of EDIT

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

```
{DOWN}
{D   }        [count]
```

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

Example 1

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

```
   down
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
```

END Subcommand of EDIT

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.

```
END          [SAVE  ]
             [NOSAVE]
```

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.

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

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

EXEC Subcommand of EDIT

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

FIND Subcommand of EDIT

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

```
{FIND}
{F   }          [string [position]]
```

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. This sequence of characters must be preceded by 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 quotes (apostrophes) to accomplish the same function with the FIND subcommand. The use of single quotes as delimiters for a character string is called quoted-string notation. Following are the rules for quoted-string notation for the string operand:

1. Enclose a string within single quotes; for example, 'string character'.

2. Use two single quotes to represent a single quote within a character string; for example, 'pilgrims''s progress'.

3. Use two single quotes 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 positional operand. For COBOL data sets, the starting column is calculated from the end of the six-digit line number. If you want to consider a string starting in column 8, you must specify the digit 2 for this operand. When you use this operand with the special-delimiter form of notation for string, you must separate it from the string operand with the same special delimiter as the one preceding the string operand.

Example 1

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

Known:

The sequence of characters: ELSE GO TO COUNTER

```
find xelse go to counter
```

Example 2

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

Known:

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

```
find 'la 3,break ' 10
```

FREE Subcommand of EDIT

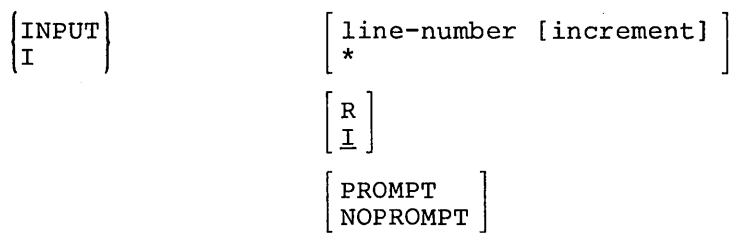
Use the FREE subcommand of EDIT to release (deallocate) previously allocated data sets that you no longer need. Refer to the FREE command for a description of the syntax and function of the FREE subcommand.

HELP Subcommand of EDIT

Use the HELP subcommand to obtain the syntax and function of EDIT subcommands. Refer to the HELP command for a description of the syntax and function of the HELP subcommand.

INPUT Subcommand of EDIT

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



line-number
specifies the line number and location for the first new line of input. If no operands are specified, input data 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
specifies that you want the system to display either a line number or, if the data set is not line numbered, a prompting character before each new input line. If you omit this operand, the default is:

- The value (either PROMPT or NOPROMPT) that was established the last time you used input mode.
- PROMPT, if this is the first use of input mode and the data set has line numbers.

- NOPROMPT, if this is the first use of input mode and the data set does not have line numbers.

NOPROMPT
specifies that you do not want to be prompted.

Example 1

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

Known:

The data set is to contain line numbers.
Prompting is 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
```

INSERT Subcommand of EDIT

Use the INSERT subcommand to insert one or more new lines of data into the data set. Input data is inserted following the location pointed to by the 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.

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

---

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.

Example 1

Operation: Insert a single line into a data set.

Known:

The line to be inserted is:

```
UCBLFG DS AL1 CONTROL FLAGS
```

The data set is not line-numbered.
The location for the insertion follows the 71st line in the data set.
The current line pointer points to the 74th line in the data set.
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.

Insert/Replace/Delete Function of EDIT

The insert/replace/delete function lets you insert, replace, or delete a line of data without specifying a subcommand name. To insert or replace a line, 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.

```
{line-number}        [string]
{    *     }
```

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.

Example 1

Operation: Insert a line into a data set.

Known:

The number to be assigned to the new line: 62
The data: (OPEN INPUT PARTS-FILE)

```
62 open input parts-file
```

Example 2

Operation: Replace an existing line in a data set.

Known:

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

```
287 go to hourcount;
```

Example 3

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

Known:

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

```
* 58 print using 120,s
```

Example 4

Operation: Delete an entire line.

Known:

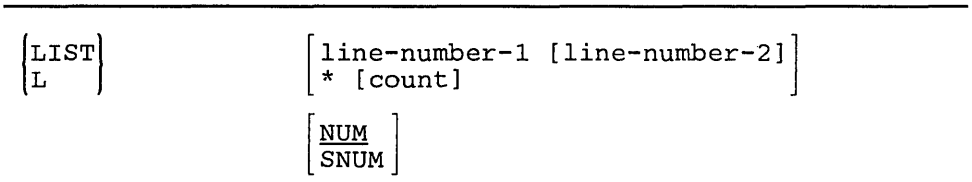
The number of the line: 98
The current line pointer in the system points to line 98.

```
 98
or
 *
```

LIST Subcommand of EDIT

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

If you do not specify any operand with LIST, the entire data set is displayed.



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
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.

Example 1

Operation: List an entire data set.

```
list
```

Example 2

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

Known:

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

```
list 27 44
```

Example 3

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

Known:

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

```
list * 17
```

MOVE Subcommand of EDIT

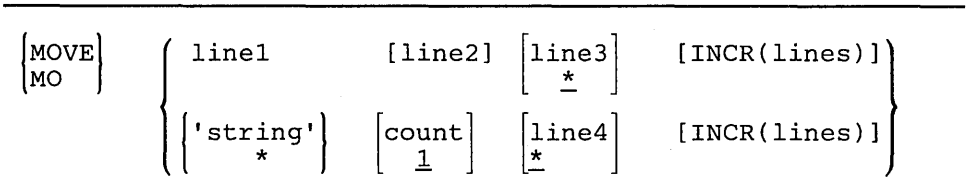
Use the MOVE subcommand of EDIT to move one or more records that exist in the data set being edited. The move operation moves data from a source location to a target location within the same data set and deletes the source data. Existing lines in the target area are shifted toward the end of the data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

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

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

If you do not specify any operand with MOVE, the data set 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.



line1

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

line2

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

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

line3

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

The value of line3 should not fall in the range from line1 to line2; that is, the target line must not be in the range being moved. Exception: Line3 can be equal to line2.

*****

represents the value of the current line pointer.

INCR(lines)

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

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

'string'

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

count

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

line4

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

Messages

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

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

1. Entering

   ```
   move * * *
   ```

   causes:

   ```
   IKJ52579I INVALID OPERANDS * INVALID FOR COUNT OR
   END OF RANGE SPECIFICATION
   ```

2. Entering

   ```
   move 100000 *
   ```

   causes:

   ```
   IKJ52579I INVALID OPERANDS FIRST LINE TO BE
   MOVED/COPIED DOES NOT EXIST
   ```

3. Entering

   ```
   move 'xyz' *
   ```

   causes:

   ```
   IKJ52579I INVALID OPERANDS QUOTED STRING NOT FOUND
   ```

4. Entering

   ```
   move 20 to 10 *
   ```

   causes:

   ```
   IKJ52579I INVALID OPERANDS END OF RANGE MUST BE
   GREATER THAN OR EQUAL TO THE BEGINNING OF THE
   RANGE
   ```

5. Entering

```
move 20 '*' 100
```

causes:

```
IKJ52579I INVALID OPERANDS STRING INVALID FOR END OF
RANGE SPECIFICATION
```

6. Entering

```
move * 0 100
```

causes:

```
IKJ52579I INVALID OPERANDS 0 INVALID FOR COUNT
```

7. Entering

```
move 10 40 20
```

causes:

```
IKJ52579I INVALID OPERANDS TRYING TO MOVE/COPY
INTO LINE RANGE
```

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

Example 1

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

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

```
Before:              Enter:                  After:

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

Note: MOVE is ignored without operands.

Example 2

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

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

```
Before:          Enter:                 After:

A                move   *  1  *        A
BB                                     BB
CCC              or                    CCC
DDDD                                   DDDD
EEEEE            move   *  1           EEEEE
FFFFFF                                 FFFFFF
GGGGGGG          or              CLP   GGGGGGG
HHHHHHHH                               HHHHHHHH
IIIIIIIII        move   *              IIIIIIIII
JJJJJJJJJJ                             JJJJJJJJJJ
KKKKKKKKKK       or                    KKKKKKKKKK
LLLLLLLLLLL                            LLLLLLLLLLL
                 move 'gg'
```

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

Example 3

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

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

```
Before:          Enter:                 After:

0010   A           move   60  50        0010   A
0020   BB                               0020   BB
0030   CCC                              0030   CCC
0040   DDDD                             0040   DDDD
0050   EEEEE                            0050   FFFFFF
0060   FFFFFF                    CLP    0060   EEEEE
0070   GGGGGGG                          0070   GGGGGGG
0080   HHHHHHHH                         0080   HHHHHHHH
0090   IIIIIIIII                        0090   IIIIIIIII
0100   JJJJJJJJJJ                       0100   JJJJJJJJJJ
0110   KKKKKKKKKK                       0110   KKKKKKKKKK
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' 99999999    THE TIME
FOR ALL                                 FOR ALL
GOOD MEN                                TO COME
TO COME                                 TO THE
TO THE                                  AID OF
AID OF                                  THEIR
THEIR                                   COUNTRY
COUNTRY                           CLP   GOOD MEN
```

Example 5

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

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

```
Before:              Enter:                          After:

0010   A             move  10  30  100  incr(5)     0040   DDDD
0020   BB                                            0050   EEEEE
0030   CCC           or                              0060   FFFFFF
0040   DDDD                                          0070   GGGGGGG
0050   EEEEE         move   9  31  100  incr(5)     0080   HHHHHHHH
0060   FFFFFF                                        0090   IIIIIIIII
0070   GGGGGGG       or                              0100   JJJJJJJJJJ
0080   HHHHHHHH                                      0105   A
0090   IIIIIIIII     move   1  39  100  incr(5)     0110   BB
0100   JJJJJJJJJJ                            CLP     0115   CCC
0110   KKKKKKKKKKK                                   0116   KKKKKKKKKKK
0120   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 through 120; source range is line 70 through line 110; target location is line 50; increment to be 10.

```
Before:             Enter:                        After:

0010   A            move  70  110  50  incr(10)  0010   A
0020   BB                                         0020   BB
0030   CCC                                        0030   CCC
0040   DDDD                                       0040   DDDD
0050   EEEEE                                      0050   EEEEE
0060   FFFFFF                                     0060   GGGGGG
0070   GGGGGGG                                    0070   HHHHHHH
0080   HHHHHHHH                                   0080   IIIIIII
0090   IIIIIIIII                                  0090   JJJJJJJJ
0100   JJJJJJJJJ                            CLP   0100   KKKKKKKKK
0110   KKKKKKKKKK                                 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   JJJJJJJJJ
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   JJJJJJJJJ                                  0107   GGGGGGG
0110   KKKKKKKKKK                                 0110   KKKKKKKKK
0120   LLLLLLLLLL                                 0120   LLLLLLLLLL
```

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   JJJJJJJJJ
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   JJJJJJJJJ                                       0158   GGGGGGG
0110   KKKKKKKKKK                                      0159   KKKKKKKKKK
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   JJJJJJJJJ
                       set.                            0151   A
                                                       0152   BB
                                                       0153   CCC
                                                       0154   DDDD
                                                       0155   EEEEE
                                                       0156   FFFFFF
                                                       0157   GGGGGGG
                                                       0158   KKKKKKKKK
                                                       0159   LLLLLLLLLLL
```

PROFILE Subcommand of EDIT

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

RENUM Subcommand of EDIT

Use the RENUM subcommand to:

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

If the data set being edited contains fixed-length records, new line numbers are placed in the last eight 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 above is overlaid.

If variable-length records without sequence numbers are being edited, 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 being edited.

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

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

---

new-line-number
specifies the new line number to be assigned to the first line renumbered. If this operand is omitted, the first line number 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.

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.

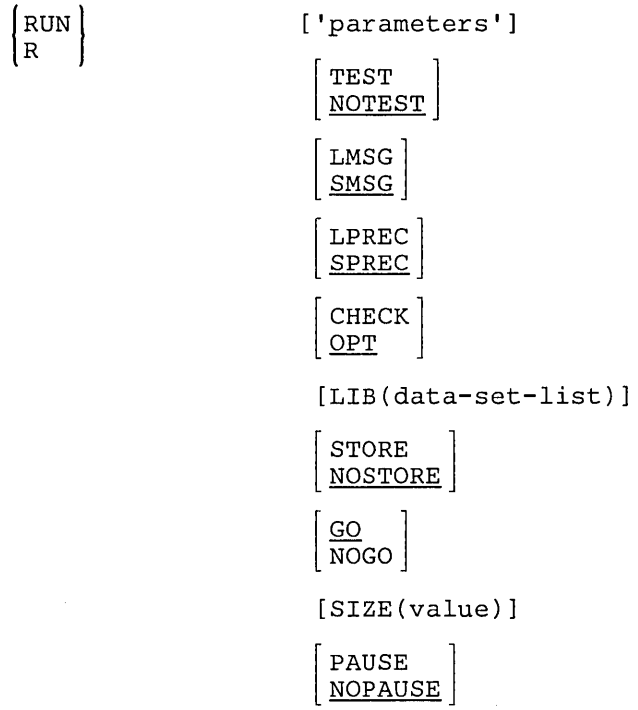
```
ren 22 2 23 29
```

RUN Subcommand of EDIT

Use the RUN subcommand to compile, load, and execute the source statements in the data set that you are editing. The RUN subcommand is designed specifically for use with certain program products. The RUN subcommand selects and invokes the particular program product 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 prior to any of the other keyword 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
specifies testing is to be performed during execution. This operand is valid for the VSBASIC program product only.

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

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
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
specifies the PL/I Checkout compiler. This operand is valid for the PL/I program product only. If you omit this operand, the OPT operand is the default value for data sets having the PLI descriptive qualifier.

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

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

STORE
specifies 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
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
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.

Example 1

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

Known:

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

```
run '1024,payroll'
```

Example 2

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

Known:

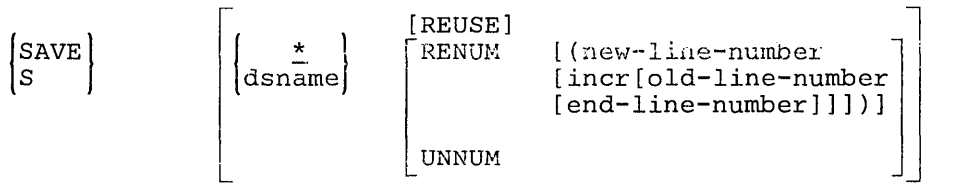
The assembler language subroutine in load module form resides in a library called USERID.MYLIB.LOAD.

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

SAVE Subcommand of EDIT

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

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.



*

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 a 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

specifies the data set specified in the DSNAME operand is to be reused if it already exists. You are not prompted for it.

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

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

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.

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.

Example 1

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

Known:

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

```
save index
```

Example 2

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

Known:

new-line-number 100
increment(INCR) 50

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

SCAN Subcommand of EDIT

Use the SCAN subcommand to request syntax checking services for statements that 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.

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

              [ON ]
              [OFF]
```

---

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

line-number-2
specifies 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
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.

Example 1

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

```
scan on
```

Example 2

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

```
scan
```

Example 3

Operation: Have several statements checked for proper syntax.

Known:

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

```
scan 62 69
```

Example 4

Operation: Check several statements for proper syntax.

Known:

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

```
scan * 7
```

SEND Subcommand of EDIT

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

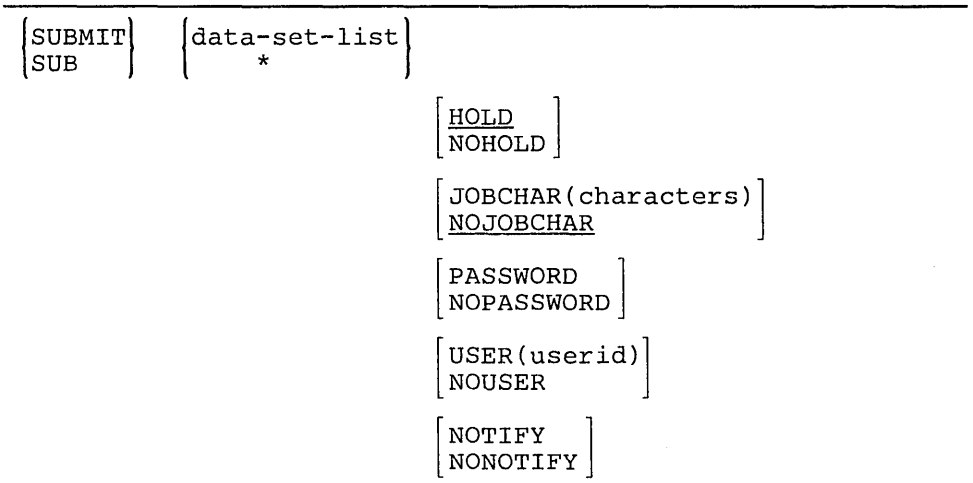
SUBMIT Subcommand of EDIT

Use the SUBMIT subcommand of EDIT to submit one or more batch jobs for 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 lower case, do not submit data sets with descriptive qualifiers TEXT or PLI.

Job cards 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 card(s) being submitted.

You must be authorized by installation management to use SUBMIT.



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

*

An asterisk (*) specifies the data set being edited 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
specifies SUBMIT has job output 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. If neither HOLD nor NOHOLD is specified, then the default is NOHOLD.

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 one 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. If neither JOBCHAR or NOJOBCHAR is specified, then the default is NOJOBCHAR.

PASSWORD
indicates a PASSWORD operand is to be inserted on the generated JOB statement by SUBMIT if the RACF program product is installed in your system. 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 the RACF program product is not installed in your installation. If the RACF program product is installed in your system, 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 being submitted. 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 the RACF program product is not installed in your system, NOPASSWORD is the default.

USER(userid)
specifies a USER operand is to be inserted on the generated JOB statement, if the RACF program product is installed in your system. 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 if the RACF program product is installed in your system, 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 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.

NOUSER
specifies generated JOB statements do not include USER and PASSWORD operands. If USER is not specified and the RACF program product is not installed on your system, then NOUSER is the default.

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=userid 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 above types of data sets containing two or more jobs is submitted for processing, certain conditions apply:

- The SUBMIT processor builds a job card for the first job in the first data set, if none is supplied, but does not build job cards for any other jobs in the data set(s).

- If the SUBMIT processor determines that the first job contains an error, none of the jobs are submitted.

- Once the SUBMIT processor submits a job for processing, errors occurring in the execution of that job have no effect on the submission of any remaining job(s) in that data set.

Any job card you supply should have a job name consisting of your 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 card, 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.

Once 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).

Example

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

Known:

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

```
submit * nonotify
```

TABSET Subcommand of EDIT

Use the TABSET subcommand to:

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

The basic form of the subcommand causes each strike of the tab key to be translated into blanks corresponding to the column requirements for the data set type. For example, if the name of the data set being edited has FORT as a descriptive qualifier, the first tabulation setting is in column 7. The values in Figure 8 is in effect when you first enter the EDIT command.

Data Set Name Descriptive Qualifier	Default Tab Settings Columns
ASM	10,16,31,72
CLIST	10,20,30,40,50,60
CNTL	10,20,30,40,50,60
COBOL	8,12,72
DATA	10,20,30,40,50,60
FORT FORTRAN(H) compilers, FORTRAN IV (G1) product data set types.	7,72
PLI PL/I Checkout and Optimizing compiler data set types.	5,10,15,20,25,30,35,40,45,50
TEXT	5,10,15,20,30,40
VSBASIC	10,15,20,25,30,35,40,45,50,55
User-defined	10,20,30,40,50,60

Figure 8. Default Tab Settings

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 (as well as 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.

```
{TABSET}     [ON [(integer-list)]]
{TAB   }     [OFF                ]
             [IMAGE              ]
```

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).

Example 1

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

Known:

Tab settings are not in effect.

```
tab
```

Example 2

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

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

Example 3

Operation: Establish tabs at every 10th column.

```
tab image
123456789t123456789t123...
```

TOP Subcommand of EDIT

Use the TOP subcommand to change the line pointer in the system to zero, that is, the pointer 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.

```
TOP
```

Example 1

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

Known:

The data set is not line-numbered.

```
top
```

UNNUM Subcommand of EDIT

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

```
{UNNUM}
{UNN  }
```

Example 1

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

Known:

The data set has line numbers.

```
unnum
```

UP Subcommand of EDIT

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

```
UP              [count]
```

---

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

Example 1

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

```
up
```

Example 2

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

```
up 17
```

VERIFY Subcommand of EDIT

Use the VERIFY subcommand to display the line that is currently pointed to by the line pointer in the system whenever the current line pointer has been moved, or whenever a line has been modified by use of the CHANGE subcommand. Until you enter VERIFY, you do not have verification of changes in the position of the current line pointer.

```
{VERIFY}        [ON ]
{V     }        [OFF]
```

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.

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.

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
```

EXEC Command

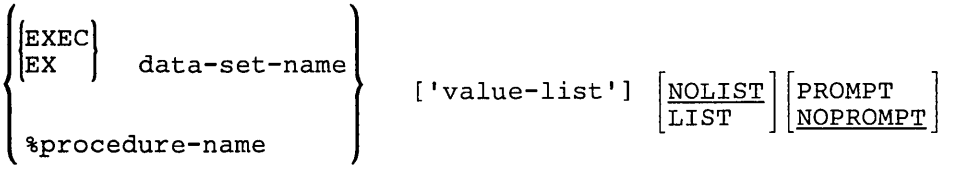
Use the EXEC command to execute a CLIST. For more information on this command and the writing of CLISTs, see *CLISTs: Implementation and Reference.*

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

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

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.

A CLIST, executed with the EXEC subcommand of EDIT, can only execute CLIST statements and EDIT subcommands.



data-set-name
specifies the name of the data set containing the CLIST to be executed. If the descriptive qualifier for the data set is not CLIST, you must enclose the fully-qualified name within apostrophes. Variable-blocked records are recommended, although you can also use fixed-blocked records. A variable-blocked record can have line numbers in the first eight columns. A fixed-blocked record can have line numbers in the last eight columns.

%procedure-name
specifies a member of a CLIST library. If the percent sign (%) is entered, TSO searches only the SYSPROC file for the specified name. Do not put quotes around the member name.

value-list

specifies the actual values that are to be substituted for the symbolic values in the CLIST. The symbolic values are defined by the operands of the PROC statement in the CLIST. The actual values to replace the *positional operands* in the PROC statement must be in the same sequence as the positional operands. The actual values to replace the *keywords* in the PROC statement must follow the positional values, but can be in any sequence. A keyword defined on the PROC statement can have a value consisting of a character string with delimiters, provided that the character string is enclosed in quotes. When you use the explicit form of the command, the value list must be enclosed in apostrophes. If apostrophes appear within the list, then you must provide two apostrophes in order to print one. If a quoted string appears as the value of a keyword within the value list, the number of quotes must be doubled again (see Example 3).

NOLIST

specifies commands and subcommands are not to be listed at the terminal. The system assumes NOLIST for implicit and explicit EXEC commands. NOLIST is the default.

LIST

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

PROMPT

specifies prompting to the terminal is allowed during the execution of a CLIST. 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 CLIST it invokes, PROMPT must be specified on a nested EXEC command.

NOPROMPT

specifies no prompting during the execution of a CLIST. NOPROMPT is the default.

No prompting is allowed during the execution of a CLIST if the NOPROMPT keyword operand of PROFILE has been specified, even if the PROMPT option of EXEC has been specified.

The following is a list of options resulting from specific keyword entries:

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

Suppose the following 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 would specify the implicit form:

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

Note: If the value of a keyword operand is not entered on the EXEC statement, that value is nullified.

Example 1

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

Note: 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 would be similar to:

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

Example 2

Operation: Assume that the CLIST in Example 1 has been stored in a CLIST library, which was allocated to the SYSPROC file ID. Execute 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 3

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 datasets MASTER.BACKUP and USER33.GOODCOPY, you could 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.

FREE Command

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

There is a maximum number of data sets that 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.

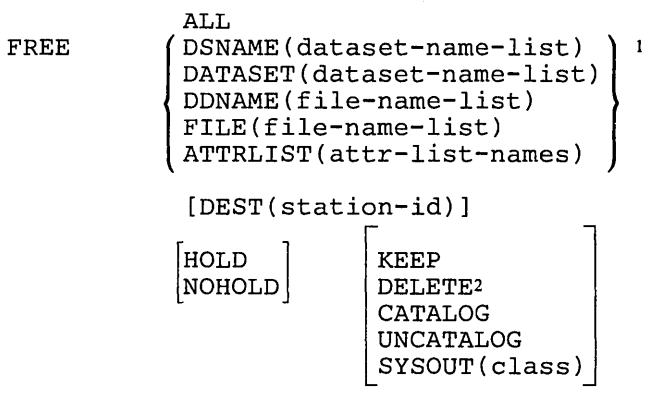
When a SYSOUT data set is freed, it is immediately available for output processing, either by the job entry subsystem (not-held data sets) or by the OUTPUT command (held data sets).

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.



1 Choose one or more of these operands within braces.

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

ALL

requests deallocation of all dynamically allocated data sets, files, and attribute lists that are not marked in-use.

DATASET or DSNAME(data-set-name-list)

specifies one or more data set names that identify the data sets that you want to free. The data set name must include the descriptive (rightmost) qualifier and can contain a member name in parentheses. If you omit this operand, you must specify either FILE, DDNAME, or the ATTRLIST operand.

FILE or DDNAME(file-name-list)

specifies one or more file names that identify the data sets to be freed. If you omit this operand, you must specify either the DATASET or DSNAME or the ATTRLIST operand.

ATTRLIST(attr-list-names)

specifies the names of one or more attribute lists that you want to delete. If you omit this operand, you must specify either the DATASET or DSNAME or the FILE or DDNAME operand.

DEST(stationid)

specifies a name of a remote work station to which the SYSOUT data sets are directed when ready for deallocation. The station ID is a one to eight character name. If this operand is omitted on the FREE command for SYSOUT data sets, the data sets are directed to the work station specified at the time of allocation.

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

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 PDS. 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.

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

SYSOUT(class)
specifies an output class which is represented by a single character. All of the system output (SYSOUT) data sets specified in the DATASET or DSNAME and FILE or DDNAME operands are assigned to this class and placed in the output queue for processing by an output writer. In order to free a file to SYSOUT, the file must have previously been allocated to SYSOUT.

A concatenated data set that was allocated in a LOGON procedure or by the ALLOCATE command can be freed only by entering the DD name on the FILE or DDNAME operand. It can also be freed by entering FREE ALL.

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

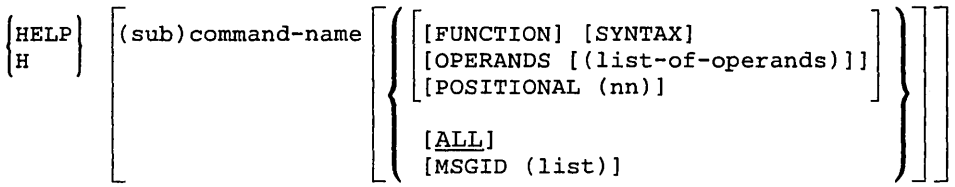
Operation: Free all dynamically allocated data sets, files, and attribute lists.

```
free all
```

HELP Command

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

You cannot use the HELP command to get additional information about CLIST statements.



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

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

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

OPERANDS(list-of-operands)
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.

POSITIONAL(nn)

specifies that you want to obtain information on 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 would be identified as '1', the second as '2', and so on. You can obtain information on the positional operands of the following commands and any of their subcommands:

- ACCOUNT
- ATTRIB
- CALL
- CANCEL
- EDIT
- EXEC
- HELP
- LOGON
- 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 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 Information: The scope of available information ranges from general to specific. The HELP command or subcommand with no operands produces a list of valid commands or subcommand and their basic functions. From the list you can select the command or subcommand most applicable to your needs. If you need more information about the selected command or subcommand, you 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 9 shows the hierarchy of the sets of information available with the HELP command or subcommand. Figure 9 also shows the form of the command or subcommand necessary to produce any particular set.

Example 1

Operation: Obtain a list of all available commands.

```
help
```

Example 2

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

```
help allocate
```

Example 3

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

```
h link operands(xref,map,coblib,ovly)
```

Example 4

Operation: Have a description of the function and syntax of the LISTBC command displayed at your terminal.

```
h listbc function syntax
```

Example 5

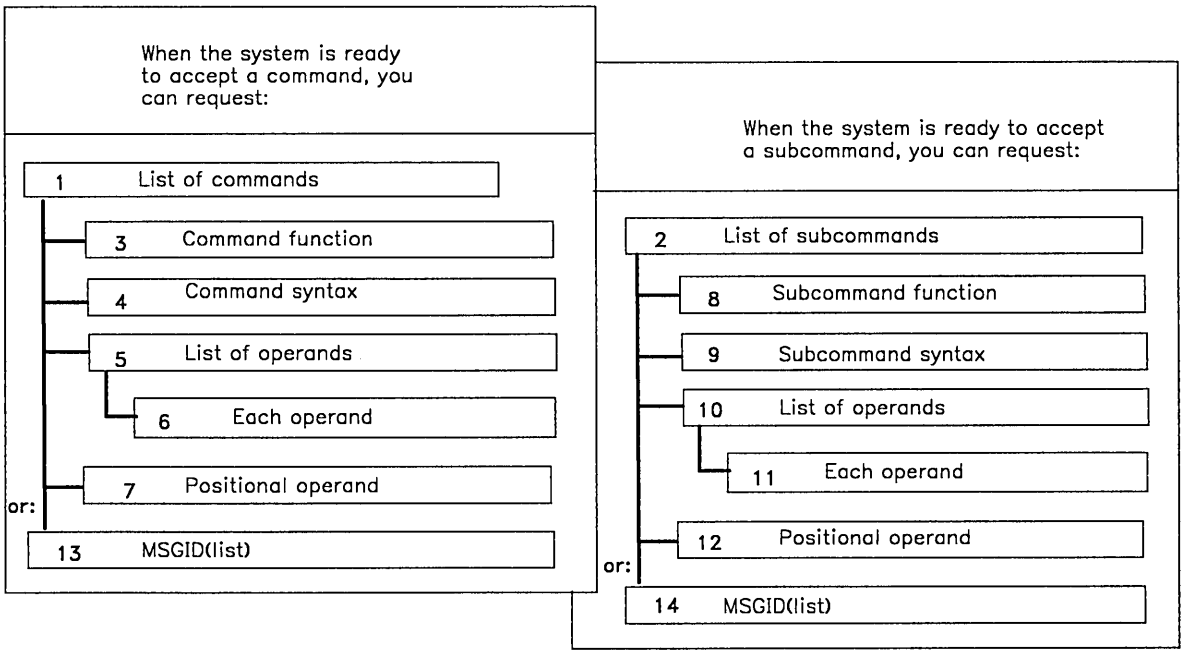
Operation: Obtain information on the ATTRIB command positional operand.

```
help attrib positional(1)
```

Example 6

Operation: Obtain information on the third positional operand of the RENUM subcommand of EDIT.

```
help renum positional(3)
```



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 9. Information Available Through the HELP Command

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

LINK Command

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

The linkage editor provides a great deal of information to help you test and debug a program. This information includes a cross-reference table and a map of the module that identifies the location of control sections, entry points, and addresses. You can have this information listed at your terminal or saved in a data set on some device.

You can specify all the linkage editor options explicitly or you can accept the default values. The default values are satisfactory for most uses. By accepting the default values, you simplify the use of the LINK command.

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 linkage editor.
- You do not want a load module.

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

```
LINK        (data-set-list)
            [LOAD [(data-set-name)]]
            [PRINT      (( * | data-set-name ))]
            [NOPRINT                           ]
            [AMODE      [(24) ]]  3
            [           [(31) ]]
            [           [(ANY)]]
            [RMODE      [(24) ]]  3
            [           [(ANY)]]
            [LIB(data-set-list)]
            [PLILIB]    [REFR  ]    [TERM  ]
            [PLICMIX]   [NOREFR]    [NOTERM]
            [PLIBASE]   [SCTR  ]      [DCBS(blocksize)]
            [FORTLIB]   [NOSCTR]
            [COBLIB]    [OVLY  ]      [AC(authorization-
                        [NOOVLY]            code)]
             [MAP  ]    [RENT  ]
             [NOMAP]    [NORENT]
             [NCAL ]  [SIZE(integer1 integer2)]
             [NONCAL]
             [LIST ]    [NE  ]
             [NOLIST]   [NONE]
             [LET  ]    [OL  ]
             [NOLET]    [NOOL]
             [XCAL ]    [DC  ]
             [NOXCAL]   [NODC]
             [XREF ]    [TEST  ]
             [NOXREF]   [NOTEST]
             [REUS ]
             [NOREUS]
```

(data-set-list)

specifies the names of one or more data sets containing your object modules and/or linkage editor control statements. 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.

3 MVS/XA only

LOAD(data-set-name)
specifies the name of the partitioned data set that contains 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.

PRINT(data-set-name or *)
specifies 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 no linkage editor listings are to be produced. This operand causes the MAP, XREF, and LIST options to become invalid. If both PRINT and NOPRINT are omitted and you do not use the LIST, MAP, or XREF operand, then NOPRINT is the default.

AMODE (MVS/XA Only)
specifies the addressing mode for the module to be link-edited. If the AMODE operand is not specified, the AMODE defaults to the AMODE of the main entry point.

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

ANY to indicate the module is to be invoked in the addressing mode of the caller

RMODE (MVS/XA Only)
specifies the residence mode for the module to be link-edited. 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:

24 to indicate the module must reside below the 16 megabyte line

ANY to indicate the module can reside anywhere in virtual storage

LIB (data-set-list)
specifies one or more names of library data sets to be searched by the linkage editor to locate object modules referred to by the module being processed; that is, to resolve external references. When you specify more than one name, the names must be separated by a valid delimiter. If you specify more than one name, the data sets are concatenated to the file name of the first data set in the list. For control statements, the first data set in the list must be pre-allocated with the DD name or file name SYSLIB prior

to the LINK command. If you specify more than one name, the data sets are concatenated to the file name of the first data set and lose their individual identity. See *System Programming Library: System Macros and Facilities* for details on dynamic concatenation.

PLILIB
specifies the partitioned data set named SYS1.PL1LIB is to be searched by the linkage editor to locate load modules that are referred to by the module being processed.

PLIBASE
specifies the partitioned data set named SYS1.PLIBASE is to be searched to locate load modules referred to by the module being processed.

PLICMIX
specifies the partitioned data set named SYS1.PLICMIX is to be searched to locate load modules referred to by the module being processed.

FORTLIB
specifies the partitioned data set named SYS1.FORTLIB is to be searched by the linkage editor to locate load modules referred to by the module being processed.

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

MAP
specifies 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. If both MAP and NOMAP are omitted, then NOMAP is the default.

NCAL
specifies the automatic library call mechanism is not to be invoked to locate the modules that are referred to by the module being processed when the object module refers to other load modules.

NONCAL
specifies the modules referred to by the module being processed are to be located by the automatic library call mechanism when the object module refers to other load modules. NONCAL is the default.

LIST
specifies a list of all linkage editor control statements is to be placed in the PRINT data set.

NOLIST
specifies a listing of linkage editor control statements is not to be produced. NOLIST is the default.

LET
specifies the output module is permitted to be marked as executable even though a severity 2 error is found. A severity 2 error indicates that execution of the output module might be impossible.

NOLET
specifies the output module be marked non-executable when a severity 2 error is found. NOLET is the default.

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 invalid exclusive calls are marked as errors. NOXCAL is the default.

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. Because the XREF operand includes a module map, both XREF and MAP cannot be specified for a particular LINK command.

NOXREF
specifies a cross-reference listing is not to be produced. NOXREF is the default.

REUS
specifies the load module is to be marked serially reusable if the input load module 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 be marked reusable. NOREUS is the default.

REFR
specifies the load module is to be marked refreshable if the input load module was refreshable and the OVLY operand was not specified.

NOREFR
specifies the load module is not to be marked refreshable. NOREFR is the default.

SCTR
specifies the load module created by the linkage editor can be either scatter loaded or block loaded. If you specify SCTR, do not specify OVLY.

NOSCTR
specifies scatter loading is not permitted. NOSCTR is the default.

OVLY
specifies the load module is an overlay structure and is therefore suitable for block loading only. If you specify OVLY, do not specify SCTR.

NOOVLY
specifies the load module is not an overlay structure. NOOVLY is the default.

RENT
specifies the load module is marked re-enterable provided the input load module was re-enterable and that the OVLY operand was not specified.

NORENT
specifies the load module is not marked re-enterable. NORENT is the default.

SIZE(integer1,integer2)
specifies the amount of real storage to be used by the linkage editor. The first integer (integer1) indicates the maximum allowable number of bytes. Integer2 indicates the number of bytes to be used as the load module buffer, which is the real storage area used to contain input and output data. If this operand is omitted, SIZE defaults to the size specified at system generation time.

NE
specifies the output load module cannot be processed again by the linkage editor or loader. The linkage editor does not create an external symbol dictionary. If you specify either MAP or XREF, then the NE operand is invalid.

NONE
specifies the output load module can be processed again by the linkage editor and loader and that an external symbol dictionary is present. NONE is the default.

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.

DC
specifies the output module can be reprocessed by the linkage editor (level E).

NODC
specifies the load module cannot be reprocessed by the linkage editor (level E). NODC is the default.

TEST
specifies the symbol tables created by the assembler and contained in the input modules are to be placed into the output module.

NOTEST
specifies no symbol table is to be retained in the output load module. NOTEST is the default.

TERM
specifies you want error messages directed to your terminal as well as to the PRINT data set. TERM is the default.

NOTERM
specifies you want error messages directed only to the PRINT data set and not to your terminal.

DCBS(blocksize)
specifies the block size of the records contained in the output load module. The block size must be an integer.

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

Example 1

Operation: Combine three object modules into a single load module.

Known:

The names of the object modules in the sequence that the modules must be in:
TPB05.GSALESA.OBJ TPB05.GSALESB.OBJ TPB05.NSALES.OBJ

You want all of the linkage editor listings to be produced and directed to your terminal.

The name for the output load module:
TPB05.SALESRPT.LOAD(TEMPNAME)

```
link (gsalesa,gsalesb,nsales) load(salesrpt) print(*) -
 xref list
```

Example 2

Operation: Create a load module from an object module, an existing load module, and a standard processor library.

Known:

The name of the object module: VACID.M33THRUS.OBJ

The name of the existing load module: VACID.M33PAYLD.LOAD(MOD1)

The name of the standard processor library used for resolving external references in the object module: SYS1.PLILIB

The entry name of the load module is MOD2.

The alias name of the load module is MOD3.

The name of the output load module: VACID.M33PERFO.LOAD(MOD2)

```
link(m33thrus,*) load(m33perfo(mod2)) print(*) -
 plilib map list
```

Choosing ld2 as a 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 (MVS/XA Only)

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 data sets allocated during the current TSO session. Included in the total number of data sets that the system allows a user to allocate dynamically are data sets that had been previously allocated for temporary use by a command.

The LISTALC command without operands produces a list of all data set names (including those that are not partitioned), which have either been allocated by you or temporarily allocated by previous TSO commands. This list includes terminal data sets, indicated by the word TERMINAL, and also attr-list-names created by the ATTRIB command, indicated by the word NULLFILE.

LISTALC displays a list of data set names allocated by the terminal user. If an asterisk precedes a data set name, it indicates that the data set is allocated, but marked not-in-use.

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

STATUS
specifies that you want information about the status of each data set. This operand provides you with:

- The data definition name (DDNAME) for the data set allocated and the attr-list-names created by the ATTRIB command.
- The scheduled and conditional dispositions of the data set. The operands denoting the dispositions are CATLG, DELETE, KEEP and UNCATLG. The scheduled disposition is the normal disposition and precedes the conditional disposition when listed. The conditional disposition takes effect if an abnormal termination occurs. CATLG means 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.

HISTORY
specifies that you want to obtain information about the history of each data set. This operand provides you with:

- The creation date
- The expiration date
- An indication as to whether or not the data set has password protection (non-VSAM only)
- The data set organization (DSORG). The listing contains:

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

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

MEMBERS
specifies that you want to obtain the library member names of each partitioned data set having your user's identification as the leftmost qualifier of the data set name. Aliases are included.

SYSNAMES
specifies that you want to obtain the fully qualified names of data sets having system-generated names.

Example 1

Operation: Obtain a list of the names of all the data sets allocated to you.

```
listalc
```

Example 2

Operation: Obtain a list of the names of all the data sets allocated to you. At the same time obtain the creation date, the expiration date, password protection, and the data set organization for each data set allocated to you.

```
lista history
```

Example 3

Operation: Obtain all available information about the data sets allocated to you.

```
lista members history status sysnames
```

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

```
listalc mem status sysnames history
--DSORG--CREATED--EXPIRES---SECURITY---DDNAME---DISP
RRED95.ASM
 PS 00/00/00 00/00/00 WRITE EDTDUMY1 KEEP
RRED95.EXAMPLE
 PO 00/00/00 00/00/00 PROTECTED EDTDUMY2 KEEP,KEEP
--MEMBERS--
 MEMBER1
 MEMBER2
SYS70140.T174803.RV000.TSOSPEDT.R0000001
 ** 00/00/00 00/00/00 NONE SYSUT1 DELETE
ALLOCATION MUST BE FREED BEFORE RESOURCES CAN BE
RE-USED
 EDTDUMY3
 SYSIN
 SYSPRINT
READY
```

Example 4

Operation: List the names of all your active attribute lists allocated with 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
```

LISTBC Command

Use the LISTBC command to obtain a listing of the contents of the broadcast data set, SYS1.BRODCAST. The SYS1.BRODCAST data set contains messages of general interest (NOTICES) that are sent from the system to all terminals and messages directed to a particular user (MAIL). The system deletes MAIL messages from the data set after they have been sent. NOTICES must be deleted explicitly by the operator.

```
{LISTBC}      [MAIL   ]
{LISTB }      [NOMAIL ]

              [NOTICES  ]
              [NONOTICES]
```

MAIL
specifies that you want to receive the messages from the broadcast data set that are intended specifically for you. If both MAIL and NOMAIL are omitted, then MAIL is the default.

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

NOTICES
specifies that you want to receive the messages from the broadcast data set that are intended for all users. If both NOTICES and NONOTICES are omitted, then NOTICES is the default.

NONOTICES
specifies that you do not want to receive the messages that are intended for all users.

Example 1

Operation: Specify that you want to receive all messages.

```
listbc
```

Example 2

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

```
listbc nomail
```

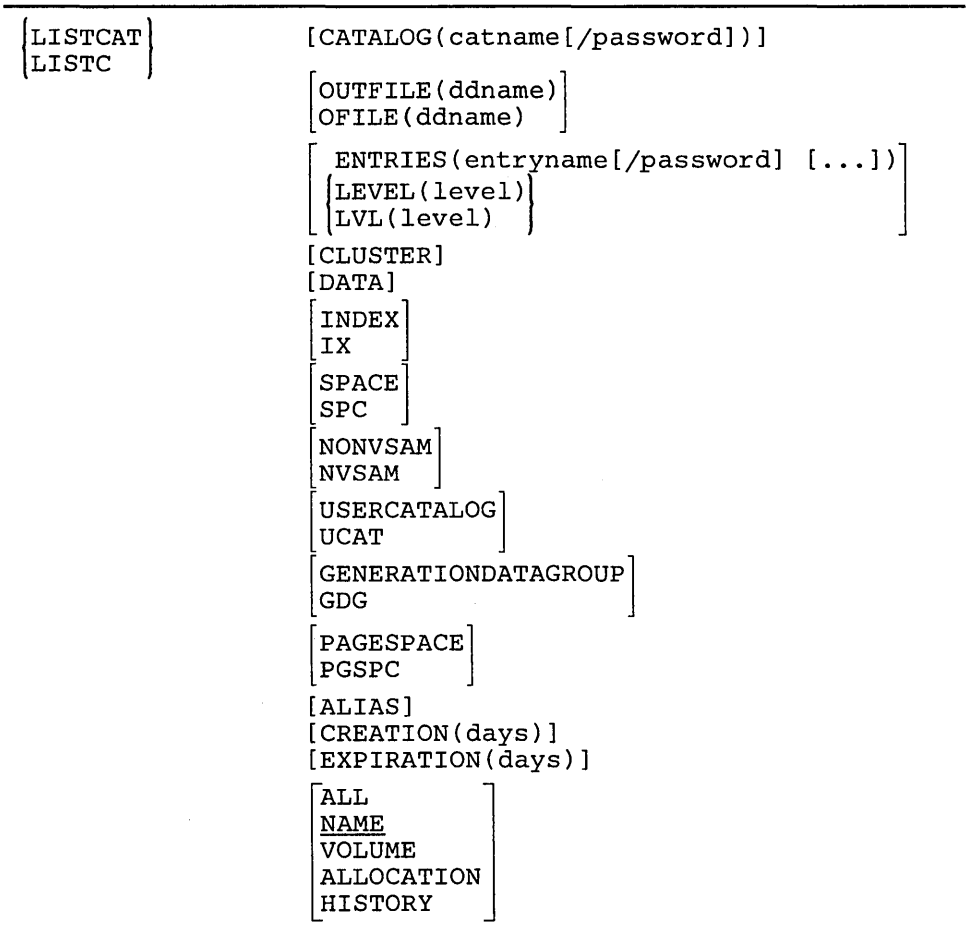
LISTCAT Command

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.

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

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

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.



CATALOG(catname[/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.

catname
is the name of the catalog.

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

OUTFILE(ddname) or OFILE(ddname)
specifies a data set other than the terminal to be used as an output data set. The DD name 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 DD name 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 operands of this data set are:

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

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

It is possible to alter the above defaults through specification of the desired values in the DCB 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(entryname[/password])
specifies the names of the entries to be listed. If neither ENTRIES nor LEVEL is coded, only the entries associated with the user ID are listed. See *Access Method Services.*

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

Note: 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) or LVL(level)
specifies the level of entry names to be listed. If neither LEVEL nor ENTRIES is coded, only the entries associated with the user 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 or 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 or SPC
specifies entries for volumes containing data spaces defined in this 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 or 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 or 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.

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

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

ALL
specifies all fields are to be listed.

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

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.

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

LISTDS Command

Use the LISTDS command to have the attributes of specific data sets displayed at your terminal. 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 print only the data set organization (DSORG), which is VSAM. Use the LISTCAT command to obtain more information on a VSAM data set.

For non-VSAM data sets, you can obtain:

- The volume identification (VOLID) of the volume on which the data set resides. A volume can be a disk pack or a drum.
- 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 *datasetname(membername)*.
- Creation date, expiration date, and, for non-VSAM only, security attributes.
- File name and disposition
- Data set control blocks (DSCB)

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

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

(data-set-list)
specifies one or more data set names. This operand identifies the data sets that you want to know more about. Each data set specified must be currently allocated or available from the catalog, and must reside on a currently active volume. The names in the data set list 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 first qualifier and C for 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 currently scheduled data set disposition and the conditional disposition. The keywords denoting the dispositions are CATLG, DELETE, KEEP, and UNCATLG. The scheduled disposition is the normal disposition and precedes the conditional disposition when listed. The conditional disposition takes effect if an abnormal termination occurs. CATLG means 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.

HISTORY
specifies that you want to obtain the creation and expiration dates for the specified data sets and find out whether or not the non-VSAM data sets are password-protected.

MEMBERS
specifies that you want a list of all the members of a partitioned data set, including 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(cat. name)
specifies the user catalog that contains the names in the data set list. CATALOG is required only if the names are in a catalog other than STEPCAT or the catalog implied by the first-level qualifier of the name.

LEVEL
specifies 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.

Example

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

Known:

The data set name: ZALD58.CIR.OBJ

```
listds cir
```

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

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

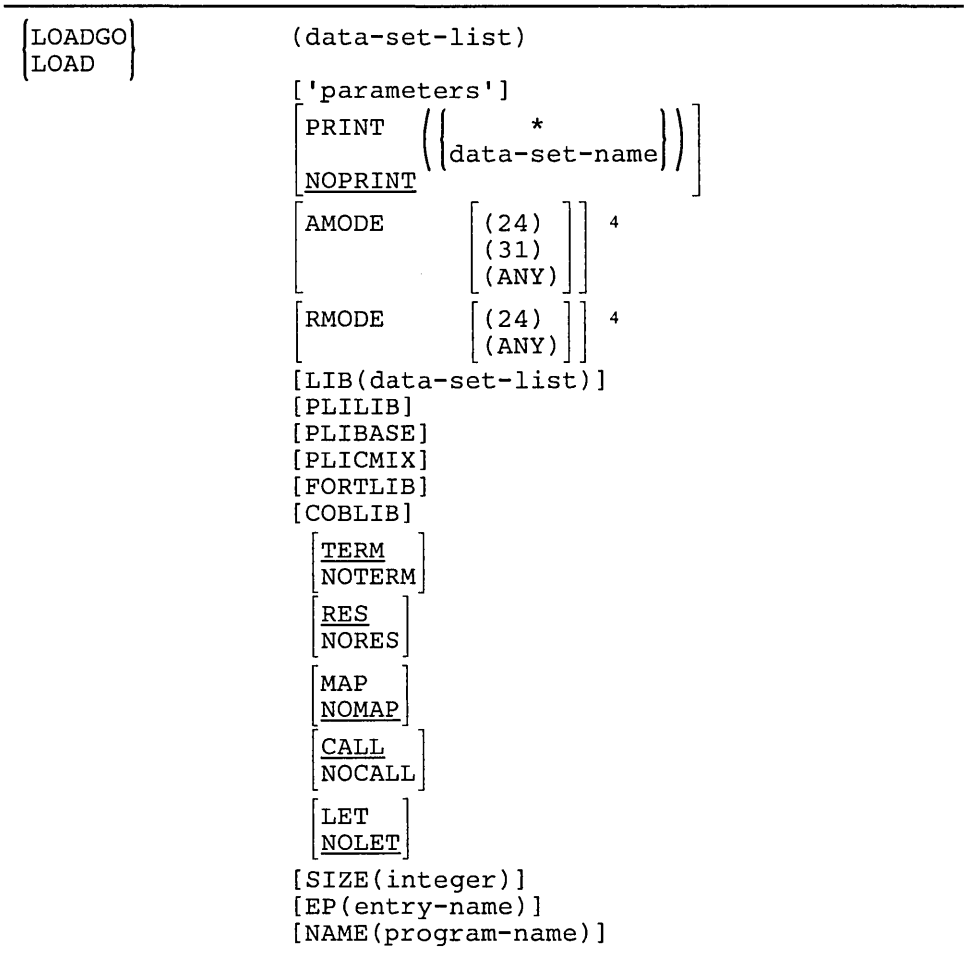
LOADGO Command

Use the LOADGO command to load a compiled or assembled program into real storage and begin execution.

The LOADGO command loads object modules produced by a compiler or assembler, and load modules produced by the linkage editor. If you want to load and execute a single load module, the CALL command is more efficient. The LOADGO command will also search a call library (SYSLIB) or a resident link pack area, or both, to resolve external references.

The LOADGO command invokes the system loader to accomplish this function. The loader combines basic editing and loading services of the linkage editor and program fetch in one job step. Therefore, the *load* function is equivalent to the *link-edit and go* function.

The LOADGO command does not produce load modules for program libraries, and it does not process linkage editor control statements such as INCLUDE, NAME, OVERLAY, etc.



4 MVS/XA only

(data-set-list)
specifies the names of one or more object modules and/or load modules 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 or *)
specifies the name of the data set that is to contain the listings produced by the LOADGO command. If you omit the data set name, the generated data set 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 (MVS/XA Only)
specifies the addressing mode for the module to be loaded. If the AMODE operand is not specified, the AMODE defaults to the AMODE of the main entry point.

Valid AMODE values are:

24 to indicate the module is invoked in 24-bit addressing mode

31 to indicate the module is invoked in 31-bit addressing mode

ANY to indicate the module is invoked in the addressing mode of the caller

RMODE (MVS/XA Only)
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 megabyte line

ANY to indicate the module can reside anywhere in virtual storage

LIB(data set list)
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).

PLILIB
specifies the partitioned data set named SYS1.PL1LIB is to be searched to locate load modules referred to by the module being processed.

PLIBASE
specifies the partitioned data set named SYS1.PLIBASE is to be searched to locate load modules referred to by the module being processed.

PLICMIX
specifies the partitioned data set named SYS1.PLICMIX is to be searched to locate load modules referred to by the module being processed.

FORTLIB
specifies the partitioned data set named SYS1.FORTLIB is to be searched to locate load modules referred to by the module being processed.

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

TERM
specifies that you want any error messages directed to your terminal as well as 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
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 invalid.

NORES
specifies the link pack area is not to be searched to locate modules referred to by the module being processed.

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
specifies the data set specified in the LIB operand is to be searched to locate load modules referred to by the module being processed. If both CALL and NOCALL are omitted, then CALL is the default.

NOCALL
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 invalid when you specify NOCALL.

LET
specifies execution is to be attempted even if a severity 2 error should occur. A severity 2 error indicates that execution might be impossible.

NOLET
specifies execution is not to be attempted if a severity 2 error should occur. If both LET and NOLET are omitted, then NOLET is the default.

SIZE(integer)
specifies the size, in bytes, of dynamic real storage that can be used by the loader. If this operand is not specified, then the size defaults to the size specified at system generation time.

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.

NAME(program-name)
specifies the name that you want assigned to the loaded program.

Example 1

Operation: Load and execute an object module.

Known:

The name of the data set: SHEPD58.CSINE.OBJ

```
load csine print(*)
```

Example 2

Operation: Combine an object module and a load module, and then load and execute them.

Known:

The name of the data set containing the object module: LARK.HINDSITE.OBJ

The name of the data set containing the load module: LARK.THERMOS.LOAD(COLD)

```
load (hindsite thermos(cold)) print(*) +
lib('sys1.sortlib') +
nores map size (44k) ep (start23) name(thermsit)
```

Example 3 (MVS/XA Only)

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 would 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.

Using LOGOFF in the Background: When the LOGOFF command is executed in the background, your TSO session is terminated normally. Any remaining commands in the input stream are ignored.

```
LOGOFF          [DISCONNECT]
                [HOLD5     ]
```

DISCONNECT
specifies the line is to be disconnected following logoff. If no operand is specified, then DISCONNECT is the default.

HOLD[5]
specifies the line is not to be disconnected following logoff.

Example 1

Operation: Terminate your terminal session.

```
logoff
```

---

5 Not supported with terminals that use VTAM.

LOGON Command

Use the LOGON command to start a terminal session. If you are not familiar with the LOGON process, see *TSO/E User's Guide.*

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 userid'. TSO 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 accepts and processes them with the exception of the current and new password fields for a RACF-defined user and the current password fields for a non-RACF defined user.

TSO 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.

Before you can use the LOGON command, your installation must provide you with certain basic information:

- Your user identification (1-7 characters) and, if required by your installation, a password (1-8 alphameric characters)
- An account number (might be optional at your installation)
- A procedure name (might be optional at your installation)

Note: If you are a RACF-defined user, your installation assigns a RACF password and a GROUP name (optional) for you.

You must supply logon information to the system by using the LOGON command and operands. The information that you enter is used by the system to start and control your time sharing terminal session. You can also use the operands to specify whether or not you want to receive messages from the system or other users.

Full screen logon:

- 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.
- Requests that you enter a password.
- Allows for modification and entry of logon parameter values. If logon command parameters were not entered on the LOGON command, you can type over existing values on the menu displayed.

- Displays RACF entry fields if RACF is installed and active and the user ID is RACF-defined.

- Allows you to enter a single TSO command of up to 80 characters in length on the LOGON menu. This command is executed *after* any command entered in the PARM field on the EXEC card of the LOGON procedure. This command is also remembered from session to session.

- 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, display 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.

Using LOGON in the Background: When the LOGON command is executed in the background, your TSO session is terminated normally. Any remaining commands in the input stream are ignored.

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

---

user-identity/password/newpassword
specifies your user identification and, if required, a valid password or new password. Your user identification must be separated from the password by a slash (/) and, optionally, one or more standard delimiters (tab, blank, or comma). Your identification and password must match the identification contained in the system's user attribute data set (UADS) if you are not RACF-defined. If you are RACF-defined, you must enter the password defined in the RACF data set as the value for password. New password specifies the password that is to replace the current password. New password must be separated from the password by a slash(/) and, optionally, one or more standard delimiters (tab, blank, or comma). The new password operand is one to eight alphameric characters in length. This operand is ignored for non-RACF-defined users. (Printing is suppressed for some types of terminals when you respond to a prompt for a password.)

ACCT(account)

specifies the account number required by your installation. If the UADS contains only one account number for the password that you specify, this operand is not required. If the account number is required and you omit it, the system prompts you for it.

For TSO, an account number must not exceed 40 characters, and must not contain a blank, tab, quotation mark, apostrophe, semicolon, comma, or line control character. Right parentheses are permissible only when left parentheses balance them somewhere in the account number.

PROC(procedure-name)

specifies the name of a cataloged procedure containing the job control language (JCL) needed to initiate your session.

SIZE(integer)

specifies the maximum region size allowed for a conditional GETMAIN during the terminal session. 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

specifies messages intended for all terminal users are to be listed at your terminal during LOGON processing. If both NOTICES and NONOTICES are omitted, then NOTICES is the default.

NONOTICES

specifies that you do not want to receive the messages intended for all users during LOGON processing.

MAIL

specifies that you want messages intended specifically for you to be displayed at your terminal during LOGON processing. If both MAIL and NOMAIL are omitted, then MAIL is the default.

NOMAIL

specifies that you do not want to receive messages intended specifically for you during LOGON processing.

PERFORM(value)

specifies the performance group to be used for the terminal session. The value must be an integer from 1-999. However, the line mode LOGON limit is 255. The default value is determined by the individual installation.

RECONNECT

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

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

OIDCARD
specifies 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.

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 real storage space requirement: 90K bytes

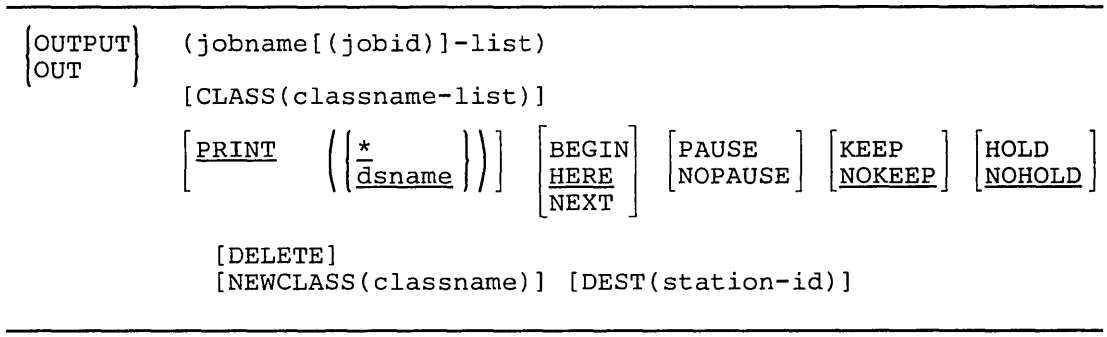
```
logon wrrid/mo@ acct(288104) proc(ts951)-
 size(90) nonotices nomail
```

OUTPUT Command

Use the OUTPUT command to:

- Direct the output from a job to your terminal. The output includes the job's job control language statements (JCL), system messages (MSGCLASS), and system output (SYSOUT) data sets.
- Direct the output from a job to a specific data set.
- Delete the output for jobs.
- Change the output class(es) for a job.
- Route the output for a job to a remote work station.
- Release the output for a job for printing by the subsystem.

OUTPUT is a foreground-initiated-background (FIB) command. This command is generally used in conjunction with SUBMIT, STATUS, and CANCEL commands.



(jobname[(jobid)]-list)
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 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.

CLASS(classname-list)
specifies the 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 or *)
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
indicates output operations for a data set are to start from the beginning of the data set whether it has been checkpointed or not.

HERE
indicates 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
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.

NOPAUSE
indicates output operations are not to be interrupted. This operand can be overridden by the PAUSE operand of the CONTINUE subcommand. If neither PAUSE nor NOPAUSE is specified, then NOPAUSE is the default.

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
specifies the kept SYSOUT data set is to be held for later access from the terminal.

Note to JES3 Users: To view output, you must specify an output class that has been defined as HOLD (for TSO) 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 *JES3 System Programming Library*.

NOHOLD
specifies the kept SYSOUT data set be released for printing by the subsystem. If neither HOLD nor NOHOLD is specified, then NOHOLD is the default for KEEP.

DELETE
specifies classes of output specified with the CLASS operand are to be deleted.

NEWCLASS(classname)
is used to change one or more SYSOUT classes to the class specified by the classname subfield.

DEST(station-id)
routes SYSOUT classes to a remote work station specified by the station ID subfield. The station ID is one to eight characters in length.

Considerations: The OUTPUT command applies to all jobs whose job names begin with your user identification. Access to jobs whose job names do not begin with a valid user identification must be provided by an installation-written exit routine. The SUBMIT, STATUS, and CANCEL commands apply to 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 in order to be available at the terminal.

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),
//              DIPS=(OLD,KEEP),
//              DCB=(RECFM=U,BLKSIZE=3036)
//SYSUT2        DD            SYSOUT=Z
//SYSIN         DD            *
              PRINT TYPORG=PS,TOTCONV=XE
              LABELS DATA=NO
/*
//JWSD582       JOB           91435,MSGCLASS=X
//              EXEC          PGM=IEHPROGM
//SYSPRINT      DD            SYSOUT=Y
//DD2           DD            UNIT=3330,VOL=SER=333000,
//              DISP=OLD
//SYSIN         DD            *
              SCRATCH VTOC,VOL=3330=333000
/*
```

To retrieve the output, you enter:

```
output (jwsd581 jwsd582) class (x y z)
```

Your output 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: 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.

Example 1

Operation: Direct the held output from a job to your terminal. Skip any checkpointed data sets.

Known:

The name of the job: SMITH2

The job is in the system output class: SYSOUT=X

Output operations are to be resumed with the next SYSOUT data set or group of system messages that have never been interrupted. You want the system to pause after processing each output data set.

```
output smith2 class(x) print(*) next pause
```

Example 2

Operation: Direct the held output from two jobs to a data set so that it can be saved and processed at a later date.

Known:

The name of the jobs: JANA JANB
The name for the output data set: JAN.AUGPP.OUTLIST

```
output (jana,janb) class(r,s,t) print(augpp)
```

Example 3

Operation: Change an output class.

Known:

The name of the job: KEAN1
The existing output class: SYSOUT = S
The new output class: T

```
output kean1 class(s) newclass(t)
```

Example 4

Operation: Delete the held output instead of changing the class (see Example 3).

```
out kean1 class(s) delete
```

Example 5

Operation: Retrieve SYSOUT data from your session at your terminal.

Known:

The TSO 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

The subcommands of the OUTPUT command are:

CONTINUE
Resumes output operations that have been interrupted.

HELP
Obtains the syntax and function of the OUTPUT subcommands.

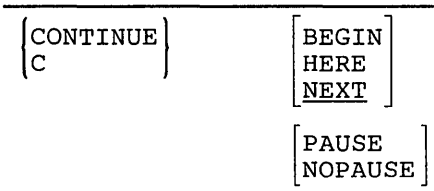
SAVE
Copies the SYSOUT data set from the spool to the named data set.

CONTINUE Subcommand of OUTPUT

Use the CONTINUE subcommand to resume output operations that have been interrupted.

Interruptions occur when:

- An output operation completes and the PAUSE operand was specified with the OUTPUT command.
- You press the attention key.



BEGIN
indicates 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/or skipped data set is 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.

Example 1

Operation: Continue output operation with the next SYSOUT data set.

```
continue
```

Example 2

Operation: Start output operations over again with the current data set being processed.

```
continue begin
```

END Subcommand of OUTPUT

Use the END subcommand to terminate the operation of the OUTPUT command.

```
END
```

HELP Subcommand of OUTPUT

Use the HELP subcommand to obtain the syntax and function of the OUTPUT subcommands. Refer to the HELP command for a description of the syntax and function of the HELP subcommand.

SAVE Subcommand of OUTPUT

Use the SAVE subcommand to copy the SYSOUT data set from the spool data set to the named data set. 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.

```
{SAVE}
{S   }        data-set-name
```

data-set-name
specifies the new data set name to which the SYSOUT data set is to be copied.

Example 1

Operation: Save an output data set.

Known:

The name of the data set: ADT023.NEWOUT.OUTLIST

```
save newout
```

Example 2

Operation: Save an output data set.

Known:

The name of the data set: BXZ037A.OLDPART.OUTLIST
The data set member name: MEM5
The data set password: ZIP

```
save oldpart(mem5)/zip
```

PROFILE Command

Use the PROFILE command or 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 or not prompting is to occur.
- Specify the frequency of prompting under the EDIT command.
- Specify whether or not you want to accept messages from other terminals.
- Specify whether or not you want the opportunity to obtain additional information about messages from a CLIST.
- Specify whether or not you want message numbers for diagnostic messages displayed at your terminal.

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 Figure 10, provide for the character-delete and the line-delete control characters, depending upon what type of terminal is involved:

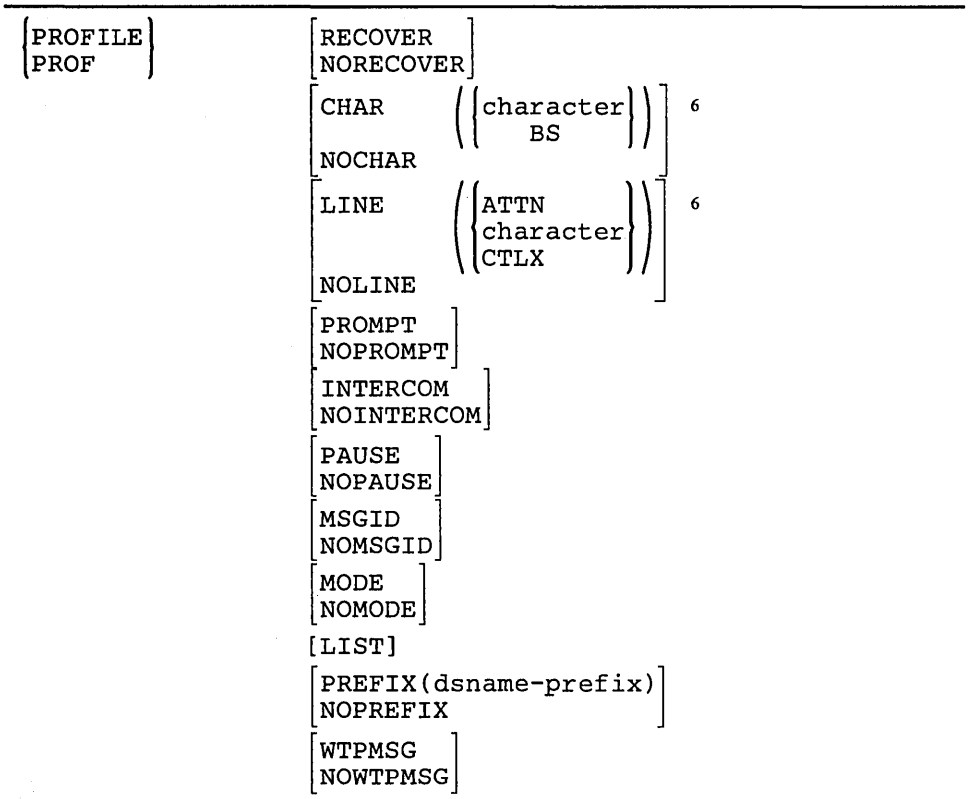
TSO Terminal	Character-Delete Control Character	Line-Delete Control Character
IBM 2741 Communication Terminal	BS(backspace)	ATTN(attention)
IBM 3270 Information Display System	None	None
IBM 3290 Information Panel	None	None
IBM 3767 Communication Terminal	None	None
IBM 3770 Data Communication System	None	None

Figure 10. System Defaults for Control Characters

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.



RECOVER
specifies that you can use the recover option of the EDIT 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)[6]
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

6 Not supported with terminals that use VTAM.

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)[7]
specifies a backspace signals that the previous character entered should be deleted. This is the default value when your user profile is created.

NOCHAR[7]
specifies no control character is to be used for character deletion.

LINE(ATTN)[7]
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 an invalid character and/or line-delete control character is entered on the PROFILE command, an error message informs you of which specific control character is invalid. 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)[7]
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)[7]
specifies the X and CCTRL keys (pressed together) on a Teletype[8] 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[7]
specifies no line-deletion control character (including ATTN) is recognized.

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

NOPROMPT
specifies no prompting is to occur.

INTERCOM
specifies that you can receive messages from other terminal users. This is the default value when your user profile is created.

---

7 Not supported with terminals that use VTAM.

8 Trademark of the Teletype Corporation.

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

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
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
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/or line-delete control character is 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)
specifies a prefix is to be appended to all non-fully qualified data set names. The prefix is composed of one-to-seven alphameric characters that begin with an alphabetic or national character.

NOPREFIX
specifies no prefixing of data set names by any qualifier is to be performed.

The default prefix in the foreground is the user ID. No prefixing of data set names is the default in the background.

WTPMSG
specifies that you want to receive all write-to-programmer messages at your terminal.

NOWTPMSG
specifies that you do not want to receive write-to-programmer messages. This is the default value when your user profile is created.

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 are not saved after the terminal sessions. However, changes you specify remain in effect for the duration of the terminal sessions. Your foreground profile is affected by background processing.

See Figure 11 for a guide to the initialization of the Terminal Monitor Program (TMP) in batch processing.

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	job user ID	NULL (blanks)
EDIT RECOV	*$	NO RECOVER	PSCBGPNM	NULL	NULL (blanks)
PROMPT	*$	NO PROMPT	OPERATOR	*	NOOPER
MSGID	*	MSGID	ACCOUNT	*	ACCOUNT
INTERCOMM	*	NO INTERCOMM	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)	SYSOUT CLASS	*	NULL (zero)
			SYSOUT DEST	*	NULL (blanks)
			CHAR DEL	*	NULL (zero)
			LINE DEL	*	NULL (zero)
			REGION SIZE	*/2	NULL (zero)
* 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 above 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.			* The value taken from the UADS entry profile.		

Figure 11. UPT/PSCB Initialization Table in the Background

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 ti&ac!bcg!.
```

and press the ENTER key, you actually enter:

```
abc.
```

PROTECT Command

Use the PROTECT command to prevent unauthorized access to your non-VSAM data set. Use the Access Method Services ALTER and DEFINE commands to protect your VSAM data set. These commands are described in *Access Method Services.*

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 would deallocate it first using the FREE command and then protect it using the PROTECT command.

```
{PROTECT}
{PROT   }      data-set-name  /control password
                [ADD (password 2)                  ]
                [REPLACE (password1   password2)   ]
                [DELETE (password1)                ]
                [LIST (password1)                  ]

               [PWREAD  ]
               [NOPWREAD]

               [PWWRITE ]
               [NOWRITE ]

               [DATA('string')]
```

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" for additional information.

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

DELETE(password1)
specifies that you want to delete an existing password, access type, or optional security information.

If the entry being removed is the control password (see the discussion following these operand descriptions), all other entries for the data set are also removed.

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

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

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

PWREAD
specifies 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
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).

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
```

Passwords

You can assign one or more passwords to a data set. Once assigned, the password for a data set must be specified in order to access the data set. A password consists of one through eight alphameric characters. You are allowed two attempts to supply a correct password.

Types of Access

Four operands determine the type of access allowed for your data set: 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.

RECEIVE Command

Use the RECEIVE command to retrieve transmitted files and to restore them to their original format.

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 ID) 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 indicator of whether the receive was successful. If an error occurred, the message number of the error is included.

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.

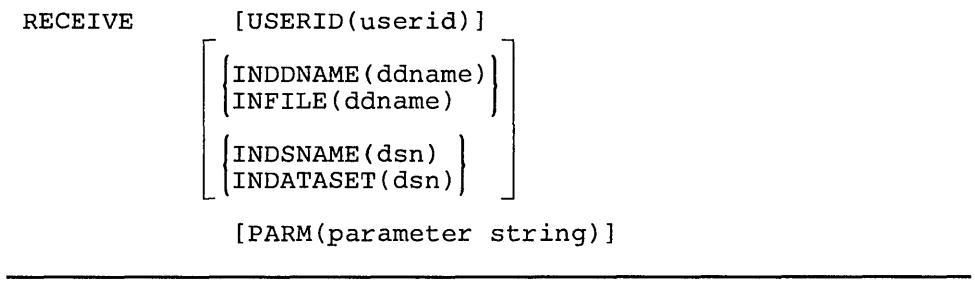
Generally, RECEIVE cannot reformat data sets. The data set into which received data is to be written must have the same record format as the original data set. The record length must be compatible. That is, equal for fixed-length records and equal or longer for variable-length records. The block size 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. The largest fixed-length record data set TSO can receive from VM is 32,760.

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, then the data set is protected. Otherwise, you should use the PROTECT command or the RACF ADDSD command to protect the 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" under the TRANSMIT command.

The RECEIVE command logs transmissions. See "Logging Function of TRANSMIT and RECEIVE" under the TRANSMIT command.

The format of the RECEIVE command is:



USERID(userid)
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) or 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 options.

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(dsn) or INDATASET(dsn)
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 options.

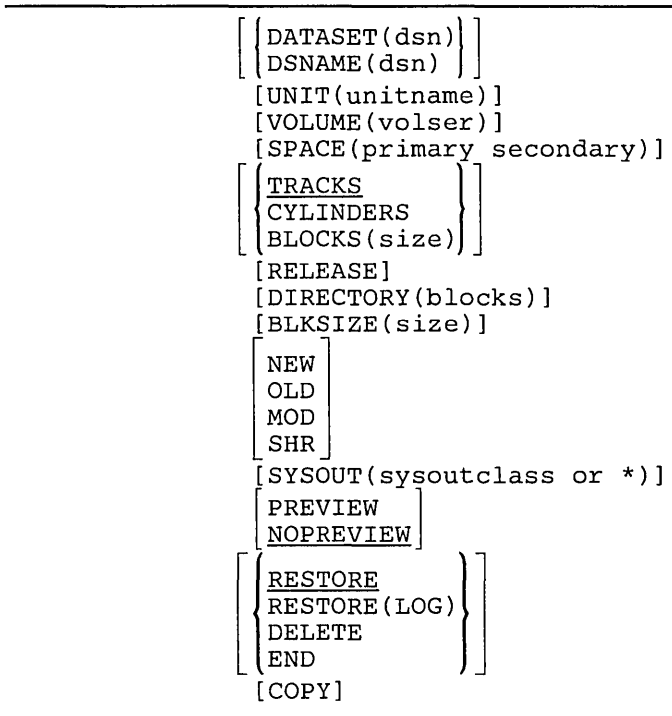
If you specify INDATASET with RECEIVE, then the transmitted data is not logged and no acknowledgment is sent to the originator. If you do not specify INDATASET, then the transmitted data is logged or written into the log entry and an acknowledgment is sent to the originator.

Use INDSNAME and INDATASET in conjunction 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.

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 as follows:



Default values for other keywords are specified with the keyword below.

DATASET(dsn)/DSNAME(dsn)
specifies the name of the data set to be used to contain the received data set. It is created if it does not already exist.

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(unitname)
specifies a unit name for a new output data set. The default value for UNIT is your normal TSO 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 2 1/2 percent of the primary.

TRACKS
specifies space to be allocated in tracks. TRACKS is the default.

CYLINDERS
specifies space to be allocated in cylinders.

BLOCKS(size)
specifies space to be allocated in blocks of the specified size.

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.

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.

SYSOUT(sysoutclass or *)
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
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
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. The DATASET or DSNAME parameter cannot be specified with RESTORE(LOG). This operand would be used 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.

Examples

In the following examples, the transmitting user is assumed to have user ID USER1 at node NODEA and the receiving user is assumed to have user ID USER2 at 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 lower case and data displayed by the system is in upper case.

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 ALLOCATION PARAMETERS, 'DELETE', OR 'END'
<null line>
RESTORE SUCCESSFUL TO DATASET 'USER2.PARMLIB'
-----------------------------------------------
No more Interactive Data Transmission Facility files
are available for the RECEIVE command to process.
```

In the above 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,ieaips00) msg 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.
<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. He terminates the receive without actually restoring the data onto the receiving system, because he does not know where he wants to store the data.

```
receive
DATASET 'SYS1.PARMLIB'  FROM USER1 ON NODEA
MEMBERS: IEASYS00, IEAIPS00
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 ALLOCATION PARAMETERS, 'DELETE', OR 'END'
end
```

In the above 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 upper and lower case just as the sender had entered it (assuming the receiver's terminal supports lower case.)

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 in order to leave space for later additions.

```
receive
DATASET 'SYS1.PARMLIB'       FROM USER1 ON NODEA
MEMBERS: IEASYS00, IEAIPS00
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 ALLOCATION PARAMETERS, 'DELETE', OR 'END'
da('nodea.parmlib') space(1) cyl dir(10)
RESTORE SUCCESSFUL TO DATASET 'NODEA.PARMLIB'
------------------------------------------------
No more Interactive Data Transmission Facility files
are available for the RECEIVE command to process.
```

The received members IEASYS00 and IEAIPS00 are 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 following text is entered on successive lines>
<of a full-screen data area.                      >
Davy,
   I sure would like to have a coonskin cap like
yours.
<Use PF3 to cause message to be sent>
```

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. A full screen input area is displayed. In this area, the user can type data, scroll using program function (PF) keys PF7 or PF19 and PF8 or PF20, and exit using PF3 or PF15.

RENAME Command

Use the RENAME command to:

- Change the name of a non-VSAM cataloged data set
- Change the name of a member of a partitioned data set
- Create an alias for a member of a partitioned data set.

The Access Method Services ALTER command changes the name of VSAM data sets and is described in *Access Method Services.*

When a password protected data set is renamed, the data set does not retain the password. You must use the PROTECT command to assign a password to the data set before you can access it.

```
{RENAME}        old-name          new-name
{REN   }        [ALIAS]
```

---

old-name
specifies the name that you want to change. The name that you specify 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.

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:

```
rename 'userid.*.data','userid.*.text'
```

or

```
rename *.data,*.text
```

Example 2

Operation: Assign an alias SUZIE to the partitioned data set member named ELIZBETH(LIZ).

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

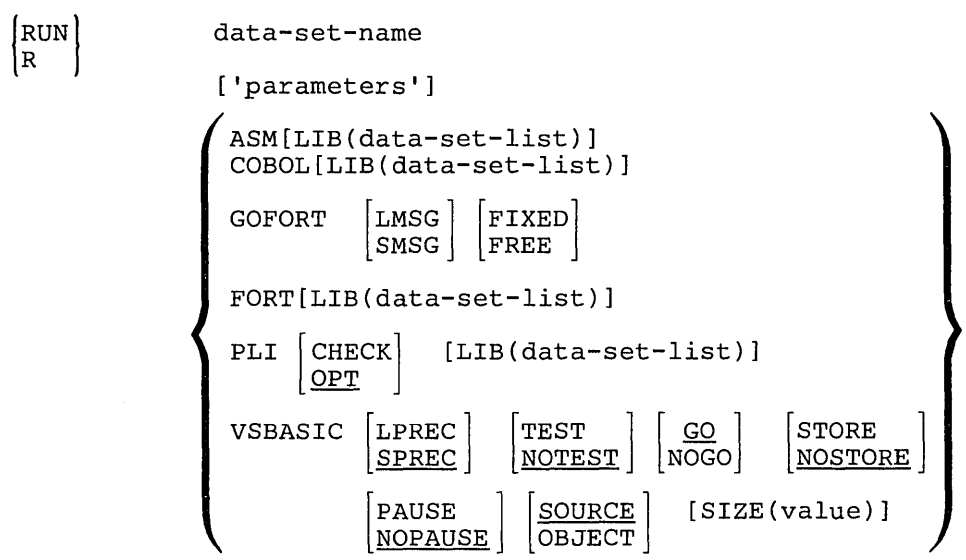
RUN Command

Use the RUN command to compile, load, and execute the source statements in a data set. The RUN command is designed specifically for use with certain program products. It selects and invokes the particular program product needed to process the source statements in the data set that you specify. Figure 12 shows which program product is selected to process each type of source statement.

Source	Program Product
Assembler	MVS/370: Assembler F and TSO Assembler Prompter MVS/XA: Assembler H Version 2
COBOL	OS/VS COBOL Release 2.4 and TSO COBOL Prompter
FORTRAN	FORTRAN IV (G1) and TSO FORTRAN Prompter VS FORTRAN
PLI	PL/I Checkout Compiler or PL/I Optimizing Compiler
VSBASIC	VSBASIC

Figure 12. Source Statement/Program Product Relationship

The RUN command and the RUN subcommand of EDIT perform the same basic function.



data-set-name 'parameters'
specifies the name of the data set containing the source program. 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
In MVS/370, ASM specifies the TSO Assembler Prompter program product and the Assembler (F) compiler are to be invoked to process source program. In MVS/XA, ASM specifies the Assembler H Version 2 compiler (without the TSO Assembler Prompter) is to be invoked to process the

source program. If the rightmost qualifier of the data set name is ASM, this operand is not required.

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

COBOL
specifies the TSO COBOL Prompter and the OS/VS COBOL program product are to be invoked to process the source program. If the rightmost qualifier of the data set name is COBOL, this operand is not required.

GOFORT
specifies the Code and Go FORTRAN program product is to be invoked to process the source program. If the right most qualifier of the data set name is GOFORT, this operand is not required.

LMSG
specifies long form diagnostic messages are to be provided.

SMSG
specifies short form diagnostic messages are to be provided.

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) program products are to be invoked to process the source program.

PLI
specifies the PL/I Prompter and either the PL/I Optimizer compiler or the PL/I Checkout compiler are to be invoked to process the source program. If the rightmost qualifier of the data set name is PLI, this operand is not required.

CHECK
specifies the PL/I Checkout compiler. If you omit this operand, the OPT operand is the default value.

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

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

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
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
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
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
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
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 thousand-byte blocks of user area where value is an integer of one to three digits.

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, 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.

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

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.

Example 1

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

Known:

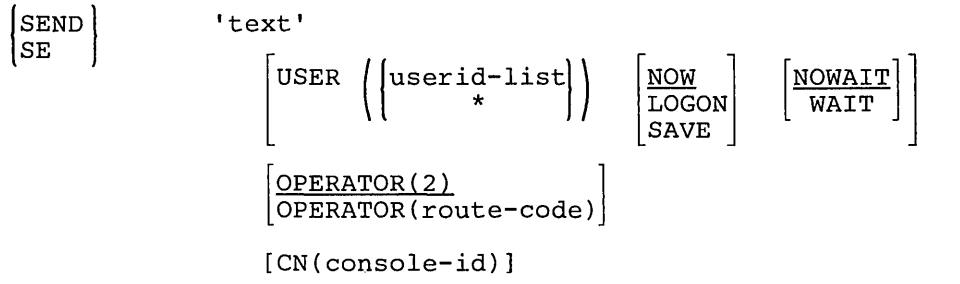
The name of the data set containing the source program is DDG39T.MNHRS.VSBASIC.

```
run mnhrs.vsbasic
```

SEND Command

Use the SEND command or SEND subcommand of EDIT to send a message to another terminal user or to the system operator. You can send a message to more than one terminal user. If the intended recipient of a message is not logged on, the message can be retained within the system and presented automatically when the recipient logs on. You are notified when the recipient is not logged on and the message is deferred.

The syntax and function of the SEND subcommand of EDIT is the same as that of the SEND command.



'text'
specifies the message to be sent. You must enclose the text of the message within apostrophes (single quotes). 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 in order to get one.

USER(userid-list)
specifies the user identification of one or more terminal users who are to receive the message. A maximum of 20 identifications can be used.

USER(*)
specifies 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 used with the SEND command at a terminal, * causes the message to be sent to the same terminal.

NOW
specifies that you want the message to be sent immediately. If the recipient is not logged on, you are notified and the message is deleted. If NOW, LOGON, and SAVE are omitted, NOW is the default value.

LOGON
specifies that you want the message retained in the SYS1.BRODCAST data set if the recipient is not logged on or is not receiving messages. When the recipient logs on, the message is removed from the data set and directed to his terminal. If the recipient is currently using the system and receiving messages, the message is sent immediately.

SAVE

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

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 for the specified users whose terminals are busy or who have not logged on. If neither WAIT nor NOWAIT is specified, 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.

OPERATOR(2) or OPERATOR(route-code)

specifies that you want the message sent to the operator indicated by the route-code. If you omit the route-code, the default is two (2); that is, the message goes to the master console operator. If both USER (identification) and OPERATOR are omitted, OPERATOR is the default value. The integer corresponds to routing codes for the write-to-operator (WTO) macro.

CN(console-id)

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

Example 1

Operation: Send a message to the master console operator.

Known:

The message: What is the weekend schedule?

```
send 'what is the weekend schedule?'
```

Example 2

Operation: Send a message to two other terminal users.

Known:

The message: If you have data set 'mylib.load' allocated, please free it. I need it to run my program.

The user identification for the terminal users: JANET5 and LYNN6

The message is important and you want to make sure the specified user gets it now.

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

Example 3

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

Known:

The recipient's user identification: BETTY7

The message: Is your version of the simulator ready?

If her terminal is busy, you might want to put the message into the SYS1.BRODCAST data set. 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 nowait
```

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}        [(jobname[(jobid)]-list)]
{ST    }
```

(jobname[(jobid)]-list)

specifies the names of the batch jobs for which you want to know the status. If two or more jobs have the same job name, the system will display the status of all the jobs encountered and supply 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. 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 (alphameric or national).

The optional job ID subfield can consist of one to eight alphameric characters (the first character must be alphabetic or national). The job ID is a unique job identifier assigned by the job entry subsystem at the time the job was submitted to the batch system.

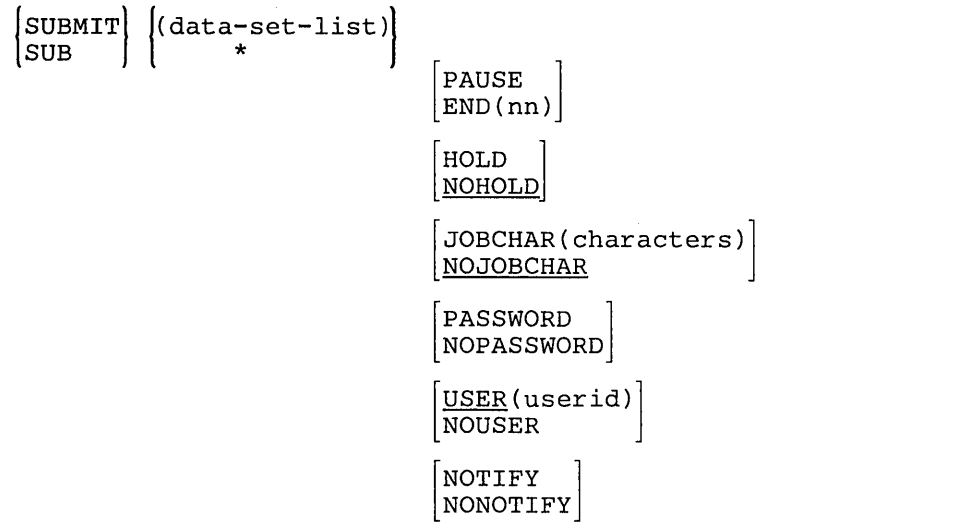
SUBMIT Command

Use the SUBMIT command to submit one or more batch jobs for background processing.

SUBMIT is a foreground-initiated-background (FIB) command. You must be authorized by installation management to use SUBMIT. This command is generally used in conjunction with the CANCEL, STATUS, and OUTPUT commands.

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.

The command syntax and description for the SUBMIT command follows:



(data-set-list)

specifies one or more data set names or names of members of partitioned data sets that define an input stream (JCL plus data). If you specify more than one data set name, 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 commands can be entered directly without creating and editing a data set. All characters in the job stream are converted to upper case prior to being processed. This positional operand and the data-set-list positional operand are mutually exclusive. This operand is required.

The SUBMIT * function is not available in EDIT mode. Job stream input received directly from the terminal or any other source will not be saved. The existing SUBMIT * function of EDIT continues to select the current

data set as the input job stream. See the SUBMIT subcommand of EDIT for more information.

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 one or two character string to indicate the end of the job stream. Only alphabetic, numeric, or national characters 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
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)
specifies characters to be appended to the jobname on every JOB statement in the data set being submitted. Use one 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
specifies a PASSWORD operand is to be inserted on the generated JOB statement by SUBMIT, if the RACF program product is installed in your system. 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 the RACF program product is not installed in

your installation. If the RACF program product is installed in your system, 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 the RACF program product is not installed in your system, NOPASSWORD is the default.

USER(userid)
specifies a USER operand is to be inserted on the generated JOB statement, if the RACF program product is installed in your system. The user ID specified is also used as the jobname for the generated JOB statement. For job name or user ID comparison for NOJBOCHAR processing, see the NOJOBCHAR operand description.

If neither USER nor NOUSER is entered and the RACF program product is installed in your system, 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.

NOUSER
specifies generated JOB statements do not include USER and PASSWORD operands. If USER is not specified and the RACF program product is not installed on your system, NOUSER is the default.

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=userid 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.

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.

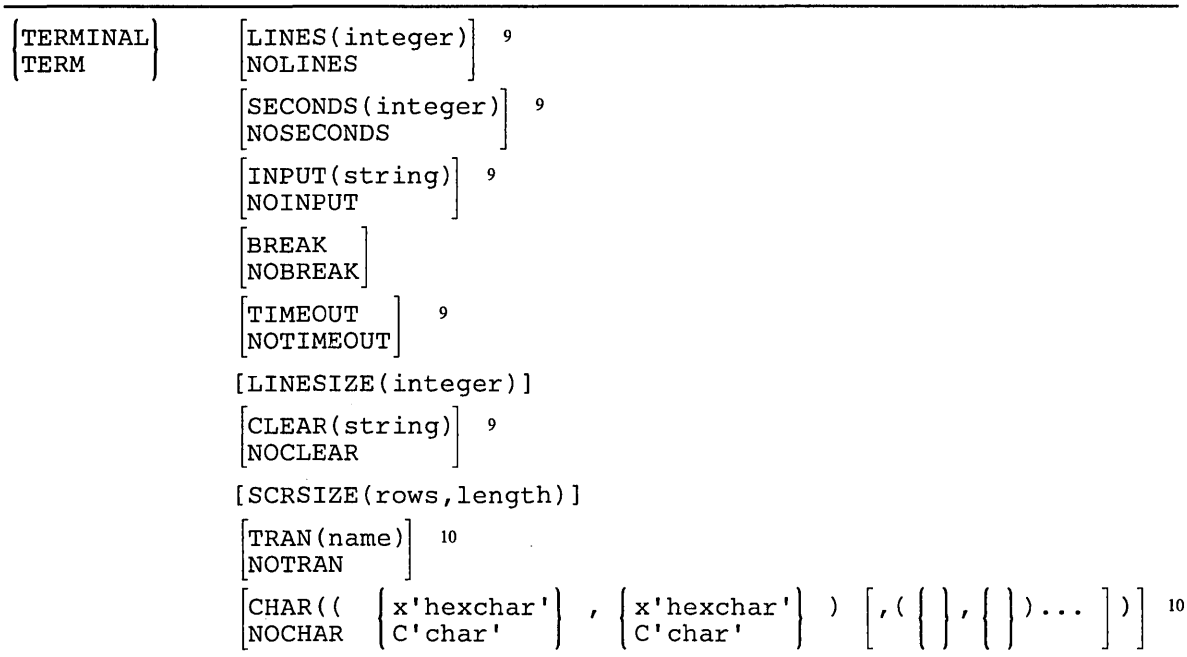
TERMINAL Command

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 or not your terminal has a key for attention interrupt.

The TERMINAL command is not allowed as a TSO 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.



9 Not supported with terminals that use VTAM.

10 Not supported with terminals that use TCAM.

LINES(integer)[11]
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[11]
specifies output line count is not to be used for controlling an attention interruption. This is the default condition.

SECONDS(integer)[11]
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[11]
specifies elapsed time is not to be used for controlling an attention interruption. This is the default condition.

INPUT(string)[11]
specifies the character string, if entered as input, will cause an attention interruption. The string must be the only input entered and cannot exceed four characters in length.

NOINPUT[11]
specifies no character string will cause an attention interruption. This is the default condition.

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 *TSO/E Guide to Writing a Terminal Monitor Program or a Command Processor.*

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.

---

[11] Not supported with terminals that use VTAM.

TIMEOUT[12]
specifies your terminal keyboard will lock automatically after approximately 9 to 18 seconds of no input.

NOTIMEOUT[12]
specifies your terminal keyboard will not lock automatically after approximately 9 to 18 seconds of no input.

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.

CLEAR(string)[12]
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 four characters in length.

NOCLEAR[12]
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 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.

---

12 Not supported with terminals that use VTAM.

'length'
specifies the maximum number of characters in a line of data displayed on the screen. Standard screen sizes are:

rows,length
6,40
12,40
12,80
15,64
24,80
27,132
32,80
43,80

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

TRAN(name)[13]
specifies a load module that contains tables used to translate specific characters you type at the terminal into different characters when they are seen by TSO. Conversely, when these characters are sent by TSO to the terminal, they are retranslated. Translation of numbers and uppercase letters is not allowed.

Character translation is especially useful when you are using a correspondence keyboard and would like to type the characters:

```
<
>
|
```

They are not available on a correspondence keyboard. Translation tables make it possible for you to specify that when you type the characters:

```
[
]
!
```

TSO interprets them as <, >, and |.

NOTRAN[13]
specifies no character translation is to take place.

CHAR[13]
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.

---

[13] Not supported with terminals that use TCAM.

NOCHAR[14]
specifies all character translations previously specified by CHAR are no longer in effect.

Example 1

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

Known:

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

```
terminal break
```

Example 2

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

Known:

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

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

---

14 Not supported with terminals that use TCAM.

TEST Command

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

If the tested program attempts to LOAD, LINK, XCTL, or ATTACH another module, the module is found according to the following search order sequence:

```
1.TASKLIB
2.STEPLIB
3.JOBLIB
4.LPA
5.LNKLST
```

If the module is not in any of these areas, it will not be found. To avoid this, bring the module into virtual storage by using the LOAD subcommand of TEST.

```
TEST            ['data-set-name']
                ['parameters']

                 [LOAD  ]
                 [OBJECT]

                 [CP  ]
                 [NOCP]
```

'data-set-name'
specifies the name of the data set containing the program to be tested. The program must be a load module that is a member of a partitioned data set or it must be an object module. A data set name must be specified to test a program that is not currently active. A currently active program is one that has abnormally terminated or has been terminated by an attention interruption.

If TEST is specified with a data set name, 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 partitioned data set, the program being tested can invoke other user load modules if they are members of the same PDS. The services by which one member can invoke another in the same PDS include LINK, LOAD, XCTL, and ATTACH.

When specifying the data set name for TEST, the name should be enclosed by single quotes or the LOAD or OBJECT qualifier is added to the name specified. If no name is specified, TSO searches for the member TEMPNAME.

Caution: The program to be tested should not have the name TEST or the name of any existing TSO service routine. For a listing of the existing module names, see *TSO/E Terminal Monitor Program and Service Routines Logic.*

'parameters'
specifies a list of parameters to be passed to the named program. The list must not exceed 100 characters, including delimiters.

LOAD
specifies the named program is a load module that has been processed by the linkage editor and is a member of a partitioned data set. 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. 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 the named program is not a command processor. NOCP is the default.

Example 1

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

Known:

Either you have received a message saying that your foreground program has terminated abnormally, or you have 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'
```

or

```
test payer.load(thrust)
```

Example 3

Operation: Invoke a program for testing.

Known:

The name of the data set that contains the program: TLC55.PAYLOAD.OBJ

The prefix in the user's profile is TLC55.

The program is an object module and is not a command processor.

```
test payload object
```

Example 4

Operation: Test a command processor.

Known:

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

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

or

```
test cmds.load(output) cp
```

Note: You will be prompted to enter a command for the command processor. TSO prompts you for the commands you 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
```

When to Use TEST

There are two basic situations in which you might use the TEST command:

1. To test a currently executing program.
2. To test a program not currently being executed.

You might want to test an executing program because it terminated abnormally or because you want to check the current environment to see that the program is executing properly.

TEST is rejected if the terminating or interrupted program is APF authorized, executing in supervisor state, or in a PSW protection key less than 8.

If a program terminates abnormally when not under TEST, you receive a diagnostic message from the terminal monitor program (TMP) followed by a READY message. If you respond to the diagnostic message with anything other than TEST, a question mark (?), or TIME, the TMP terminates your program. However, if you issue the TEST command (and supply no program name), the currently active program remains in storage when the TEST command processor gets control and you can use the TEST subcommands to debug the defective program.

You can enter both the ? and the TIME command before you issue the TEST command to debug an abnormally terminating program. However, if you want a dump, enter a null line instead of issuing the TEST command. If a SYSABEND, SYSMDUMP, or SYSUDUMP file has already been allocated, the null line results in a dump being printed.

If you want to examine the current environment of an executing program that is not terminating abnormally, enter a single attention interruption. The currently active program remains attached and the TMP responds to your interruption by issuing a READY message. When you issue the TEST command (without a program name), the currently active program remains in storage under the control of the TEST command processor. You can then use the TEST subcommands to examine the current environment.

In the case of either the ABEND or the attention interruption, you should *not* enter a program name following the TEST command. If you do, you lose the current in-storage copy of the program because TEST loads a copy of the specified program.

To test a program not currently executing, enter the TEST command supplying the data set name containing the program to be executed and any other applicable operands. When you use the TEST command to load and execute a program, the program *must* be an object module or a load module suitable for execution.

Prior to and during execution, such as when execution is interrupted at a breakpoint, you can:

- Supply initial values (test data) that you want to pass to the program

- Establish breakpoints at instructions where execution is to be interrupted so that you can examine interim results. Breakpoints should not be inserted into TSO service routines or into any of the TEST load modules.
- Display the contents of registers and virtual storage
- Modify the contents of registers and virtual storage
- Display the program status word (PSW)
- List the contents of control blocks
- Step through sections of the program, checking each instruction for proper execution.

When running in supervisor state or in a PSW protection key less than 8, breakpoints are not honored in any section of your program.

Addressing Conventions Associated with TEST

An address used as an operand for a subcommand of TEST must be one of the following types:

- **Absolute address** - a virtual storage address. An absolute address is 1 to 8 hexadecimal digits followed by a period and not exceeding X'7FFFFFFF'.
- **Relative address** - a 1 to 8 hexadecimal digit number preceded by a plus sign (+). A relative address specifies an offset from the currently qualified virtual storage address. See the discussion about qualified addresses.
- **Symbolic address** - 1 to 8 alphameric characters, the first of which is an alphabetic character. A symbolic address corresponds to a symbol in a program or a symbol defined by the EQUATE subcommand. See the discussion about qualified addresses for a detailed description of qualified symbolic addressing. See "Restrictions on the Use of Symbols" for a detailed description on the use of symbols.
- **[module-name].entry-name** - a name within a module capable of being externally referenced, preceded by a period (.), and optionally preceded by a name by which the module is known. An entry name is the symbolic address of an entry point into the module; for example, a CSECT name. A module name can be the name or alias of a load module or the name of an object module. Module or entry names can be any combination of up to eight alphameric characters, the first of which is alphabetic or national.
- **Qualified addresses** - You can qualify symbolic or relative addresses to indicate they apply to a particular module and CSECT. To do this, you must precede the address by the name of the load or object module and the name of the CSECT. The qualified address must be in the form:

```
modulename.csect.address
```

If the address is to apply to the current module, you only need to specify the CSECT name in the following form:

```
csect.address
```

If the address is to apply to the current CSECT within the current module, only the address is necessary; you do not need to qualify the address. The current module and CSECT is initially set to the program being tested. This setting is automatically changed each time a module under a different request block is invoked. This is referred to as *automatic qualification.* (This happens when a module is invoked by ATTACH, XCTL, SYNCH, or LINK. It does *not* happen when a module is loaded, called, or branched to.) The module and/or CSECT used in determining a base location for resolving symbolic and relative addresses can also be changed by using the QUALIFY subcommand.

For example, if the name of the module is OUTPUT, the CSECT is TAXES, and the symbolic address is YEAR77, you would specify either:

```
output.taxes.year77
```

or

```
.taxes.year77
```

If the current module is OUTPUT. You would specify:

```
year77
```

If the current module is OUTPUT and the current CSECT is TAXES. If the module name and CSECT name are the same as above and the address to be qualified is the relative address +4A, you would specify:

```
output.taxes.+4A
```

- **General registers** - You can refer to a general register using the AND, OR, assignment-of-value, COPY, or LIST subcommands by specifying a decimal integer followed by an R. The decimal integer indicates the number of the register and must be in the range 0 through 15. Other references to the general registers imply indirect addressing.

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

- **Floating-point registers** - You can refer to a floating-point register using the LIST or assignment-of-value subcommand by specifying a decimal integer followed by an E or D. The decimal integer indicates the number of the register and *must be* a zero, two, four, or six. An E indicates a floating-point register with single precision. A D indicates a floating-point register with double precision. The contents of the floating-point register must be assigned using the notation described in "Assignment of Values Function of TEST." You *must not* use floating-point registers for indirect addressing or in expressions.

- **Indirect address** - An address expression, a general register, or the address of a location that contains another address. An indirect address *must* be followed by one or more indirection symbols to indicate a corresponding number of levels of indirect addressing.

  The indirection symbols follow:

  - The percent sign (%), indicating that the low-order three bytes of the address are used.

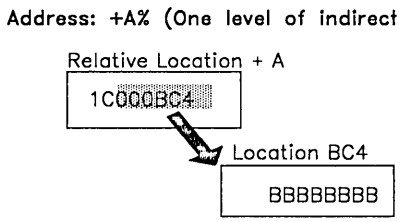
  - The question mark (?) (for **MVS/XA** only) indicating that all 31 bits are used for the address.

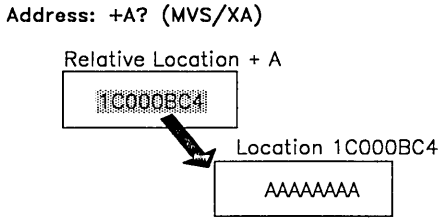
  To use a general register as an indirect address, specify a decimal integer (0 through 15) followed by an R and a percent sign, or an R and a question mark. For example, if you want to refer to data whose address is located in register 7, you would specify:

  ```
  7r%
  ```

Example: Use of a relative address to form an indirect address.

Address: +A% (One level of indirect addressing)





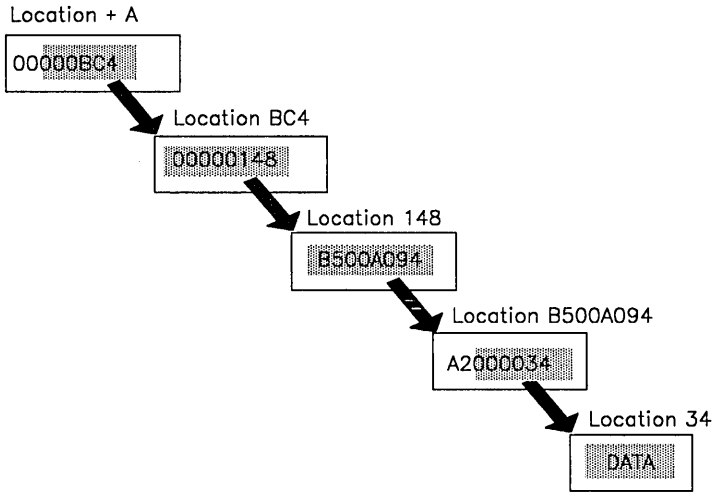
Example: Comparison of use of % and ? (**MVS/XA**)

Address	**Data**
X	0100A080
0000A080	AAAAAAAA
0100A080	BBBBBBBB

TEST Subcommand	**Data Displayed**
LIST X	0100A080
LIST X%	AAAAAAAA
LIST X? (MVS/XA)	BBBBBBBB

Example: Indirect addressing using a combination of indirection symbols.

Address expression: +A%??% (Four levels of indirect addressing)



- **Address expression** - an address followed by any number of expression values. You can specify the address as:

  - An absolute address
  - A relative address (unqualified, partially or fully qualified)
  - A symbolic address (unqualified, partially or fully qualified)
  - An indirect address.

  An expression value consists of a plus or minus displacement value expressed as either 1 to 8 hexadecimal digits or 1 to 10 decimal digits from an address in virtual storage. Following are two examples of address expressions:

  Decimal Example:

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

  ```
  ''address''
  ```

  Hexadecimal Example:

  address+14 specifies the location that is 20 decimal bytes past that

  ```
  designated by ''address''
  ```

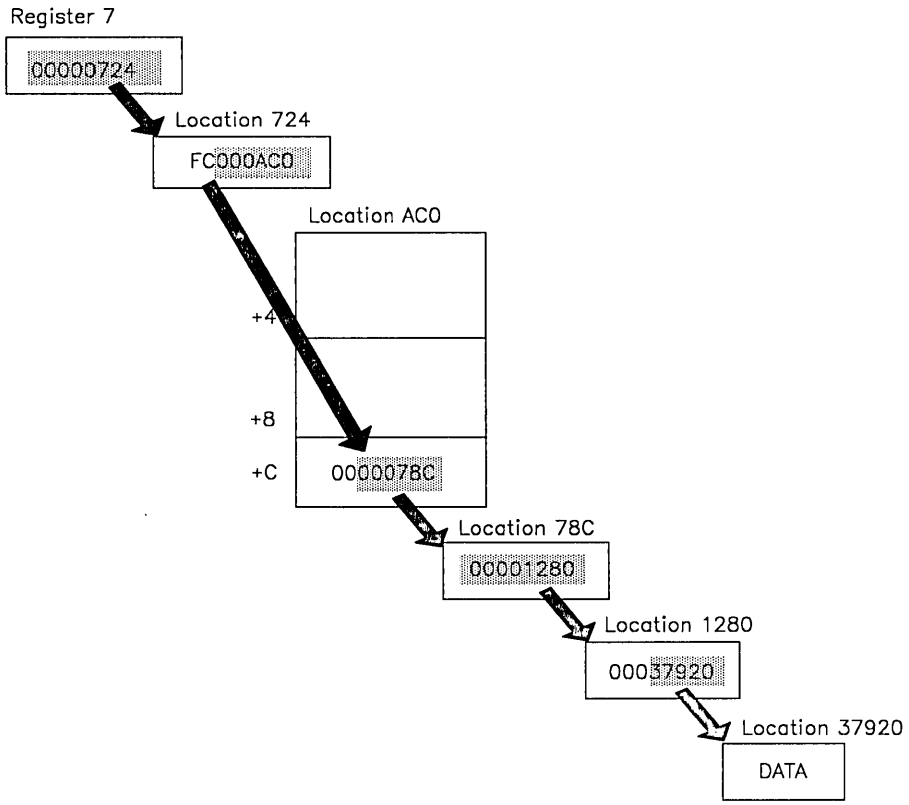
  Decimal displacement (either plus or minus) is indicated by the n following the numeric offset. You can indicate up to 256 levels of indirect addressing by following the initial indirect address with a corresponding number indirection symbols (% or ?). An address expression is specified like this:

  ```
           -          ?      -          ?
  address  +   value  % ..   +  value   %  ..  ...
  ```

If you are using an MVS/XA system, you can use any combination of percent signs and question marks after the value.

Example: Address expression with hexadecimal displacement using a combination of indirection symbols. **(MVS/XA)**

Address expression: 7R?%+C%?%



When processing an address expression, TEST checks the high-order bit of the result of each addition or subtraction. If the bit is on, indicating a negative value or overflow condition, TEST rejects the address.

Restrictions on Use of Symbols

The TEST command processor can resolve external and internal symbolic addresses, only if these addresses are available to it. Within certain limitations, symbolic addresses are available for both object modules (processed by the loader) and load modules (fetched by contents supervision).

External Symbols

The TSO TEST user can access external symbols, such as CSECT names, for a program modules, if the program was brought into main storage by the TEST command or one of its subtasks. This is the case for the program being tested, any program brought into storage through the tested program, and any program loaded by the LOAD subcommand.

External symbols for CSECT names that are in object modules are available only if the loader had enough main storage to build composite external symbol table dictionary (CESD) entries.

Internal Symbols

Internal symbols for load modules can be resolved if the CSECT containing the symbol was assembled with the TEST parameter, the module was link edited with the TEST parameter, and the program was brought into storage by the TEST command or one of its subtasks as previously explained. Names on EQU, ORG, LTORG, CNOP, and DSECT statements *cannot* be resolved.

The TSO TEST user cannot access internal symbols for object modules.

Addressing Considerations

If the necessary conditions for symbol processing are not met, you can use absolute, relative, or indirect addressing, or you can define symbols with the EQUATE subcommand of TEST.

Symbols within *DSECTs* are available only if the *DSECT* name has been defined with the EQUATE subcommand.

For example, if NAME is a symbol in a DSECT named DATATBL, then to access the data associated with NAME, you would first have to determine the address to be used as a base address for the DSECT. (This is the address in the register on the assembler USING instruction.) If the address is in register 7, you can enter:

```
equate datatbl 7r%
```

This establishes addressability to the DSECT, allowing the symbol NAME and all other symbols in the DSECT to be accessed using the symbol.

TEST can access symbols and process CSECT names (to qualify addresses and satisfy deferred breakpoints) for module loaded from a data set in LNKLIST concatenation, provided that the module was both assembled and link edited with the TEST option, and the data set involved is not READ-protected from the TSO user. Symbols and CSECT names cannot be processed for a module accessed from LPA.

Examples of Valid Addresses in TEST Subcommands

Below is a list of valid addresses which can be used with subcommands:

Address:	Type of Address:
A23C40.	Absolute
+E4	Relative
5R%	24-bit Indirect
5R? (MVS/XA only)	31-bit Indirect (MVS/XA only)
NAMES	Symbol within program
.SALES.+26	Partially-qualified relative
14R%+28	Expression
PROFIT.SALES	Module and entry name
+16+10n	Expression
.SALES.NAMES	Partially-qualified symbol
PROFIT.SALES.NAMES+8n	Expression
DATA+10	Expression
.SALES	Entry name
PROFIT.SALES.NAMES	Fully-qualified symbol
6R%+4%+12n%%	Expression
PROFIT.SALES.+C0	Fully qualified relative

Note: In the preceding addresses, PROFIT is a module name, SALES is a CSECT name, and NAMES is a symbol.

31-Bit Addressing Considerations Associated with TEST (MVS/XA Only)

- All subcommands that accept addresses can process addresses above 16 Mb, regardless of the current addressing mode of the program.
- You can use the 31-bit indirection symbol (?) on any subcommands to reference data pointed to by 31-bit addresses.
- When TEST loads and executes a program, it uses the AMODE and RMODE characteristics to determine the addressing mode at entry as well as whether the tested program is loaded above or below 16 Mb.
- The AMODE operand on the CALL, GO, and RUN subcommands can change the addressing mode of the program being tested.
- The loader, invoked by TEST when testing an object module, loads the module above or below 16 Mb based on the RMODE characteristics of the module's CSECTs. If the first CSECT is RMODE (ANY) and any other CSECTs are RMODE (24), the loader loads the module below the 16Mb line and issues a warning message.
- Input passed in Register 1 to the program being tested by register 1 (either the CPPL or input parameter list) will be below 16 Mb.
- 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.
- Specify AMODE on the CALL subcommand, if the called program should not be invoked in the current addressing mode. When control is returned, verify that the addressing mode is appropriate before continuing execution.

Programming Considerations Associated with TEST When Using the Virtual Fetch Services (MVS/XA Only)

External symbols are not available for a program fetch. For information on addressing considerations, see "Restrictions on Use of Symbols."

Do not establish deferred breakpoints for a program managed by virtual fetch because they are ignored.

If you are testing program A, which invokes program B using the virtual fetch services, you cannot use TEST subcommands to stop execution of program B.

If, while testing program A, you want to debug program B, you can use the following method. Instead of allowing a virtual fetch GET request to pass control to program B, load and call program B using TEST subcommands.

- Use the AT subcommand of TEST to establish a breakpoint immediately before the virtual fetch GET request in program A.

- When you reach the breakpoint, use the LOAD subcommand of TEST to load a different copy of program B.
- You can then establish breakpoints using the AT subcommand at any points in this copy of program B.
- Use the CALL subcommand of TEST to execute program B. Specify an address on the RETURN parameter to bypass the virtual fetch GET request in program A.

Note: You cannot use TEST facilities to debug a program's interface with virtual fetch.

See *System Programming Library: System Modifications* for a description of the virtual fetch services.

Programming Considerations Associated with TEST for Use in a Cross-Memory Environment

Attention Interruptions in Cross-Memory Mode - If an attention interrupt occurs while the program being tested is executing in cross-memory mode, and you enter anything other than a null line, the cross-memory environment is terminated and a message is displayed.

Access to Storage by TEST - If TEST is used with cross-memory applications, access to storage by TEST subcommands is restricted to the home address space.

Abend In Cross-Memory Mode - If an abend occurs while a cross-memory application is executing outside the home address space, TSO TEST does not preserve the cross-memory environment. The registers and PSW at the time of the abend and the abend code from the error message are the only debugging information available for a cross-memory abend.

Restrictions on Breakpoints - Breakpoints cannot be set for the following cross-memory instructions:

PC - Program call
PT - Program transfer
SAC - Set address space control
SSAR - Set secondary ASID

TEST Subcommands

The subcommands of the TEST command are:

ALLOCATE (MVS/XA only)
dynamically allocates the data sets required by a program intended for execution.

AND (MVS/XA only)
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 (MVS/XA only)
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 (MVS/XA only)
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 (MVS/XA only)
executes a CLIST.

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 (MVS/XA only)
invokes the linkage editor service program.

LIST
displays the contents of a virtual storage area or registers.

LISTALC (MVS/XA only)
displays a list of the names of data sets allocated during the current TSO session.

LISTBC (MVS/XA only)
displays a listing of the contents of the SYS1.BRODCAST data set, which contains messages of general interest (NOTICES) and messages directed to a particular user (MAIL).

LISTCAT (MVS/XA only)
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 (MVS/XA only)
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.

LOAD
loads a program into virtual storage for execution.

OFF
removes breakpoints.

OR (MVS/XA only)
performs a logical OR operation on data in two locations, placing the results in the second location specified.

PROFILE (MVS/XA only)
establishes, changes, or lists the user profile.

PROTECT (MVS/XA only)
prevents 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 load module.

RENAME (MVS/XA only)
changes the name of a non-VSAM cataloged data set or a member of a PDS or creates an alias for a member of a PDS.

RUN
terminates TEST and completes execution of the program.

SEND (MVS/XA only)
sends a message to another terminal user or to the system operator.

STATUS (MVS/XA only)
displays status of batch jobs at terminal.

SUBMIT (MVS/XA only)
submits one or more batch jobs for processing.

TERMINAL (MVS/XA only)
defines the operating characteristics for the terminal being used.

UNALLOC (MVS/XA only)
frees data sets under TSO TEST. Because FREE is an alias for the FREEMAIN subcommand, UNALLOC must be used 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.

For a complete description of the syntax and function of the following TEST subcommands, refer to the corresponding TSO command.

ALLOCATE	PROFILE
ATTRIB	PROTECT
CANCEL	RENAME
EXEC	SEND
LINK	STATUS
LISTALC	SUBMIT
LISTBC	TERMINAL
LISTCAT	UNALLOC(FREE)
LISTDS	

ALLOCATE Subcommand of TEST (MVS/XA Only)

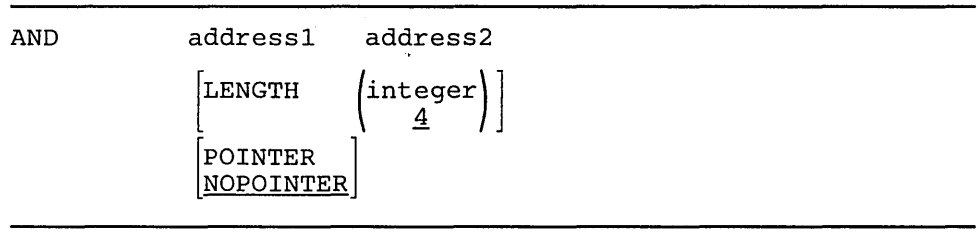
Use the ALLOCATE subcommand to dynamically allocate the data sets required by a program intended for execution. Refer to the ALLOCATE command for a description of the syntax and function of the ALLOCATE subcommand.

AND Subcommand of TEST (MVS/XA Only)

Use the AND subcommand to perform a logical AND operation on data or addresses from one virtual storage address to another, from one general register to another, from a register to virtual storage, or from virtual storage to a register.

The AND subcommand can be used to:

- Alter the contents of the general registers.
- AND an entire data field with another.



address1
specifies the location of data that is to be ANDed with data pointed to by address2.

If you do not specify POINTER and there is a breakpoint in the data pointed to by address 1, the TSO TEST processor terminates the AND operation.

address2
specifies the location of the data that is to be ANDed with data pointed to by address1. When the AND operation is complete, the result is stored at this location. You can specify address1 and address2 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).

LENGTH(integer) or 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 address1 is to be validity checked to see that it does not exceed maximum virtual storage size. Address1 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 address1.

NOPOINTER
specifies address1 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 address1, the maximum length of data that is ANDed is the total length of the general registers or 64 bytes.

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 1(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
```

Assignment of Values Function of TEST

When processing is halted at a breakpoint or before execution is initiated, you can modify values in virtual storage and in registers. This function is implicit; that is, you do not enter a subcommand name. The system performs the function in response to operands that you enter.

```
address=data-type 'value'[,data-type 'value']...
```

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.

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

Code	Type of Data	Maximum Length (Bytes)[15]	Storage Boundary Data types must begin on specified boundary for a virtual storage address
C	Character	One line of input, continued lines permitted	C-byte
X	Hexadecimal	64	X-byte
B	Binary	64	B-byte
H	Fixed point binary (halfword)	6	H-halfword
F	Fixed point binary (fullword)	11	F-fullword
E	Floating point (single precision)	13	E-fullword
D	Floating point (double precision)	22	D-doubleword
P	Packed decimal	32	P-byte
Z	Zoned decimal	17	Z-byte
A	Address constant	11	A-fullword
S	Address (base + displacement)	8	S-halfword
Y	Address constant (halfword)	6	Y-halfword

15 All characters within the quotes are included in the length.

Following is a list of valid entries and syntax for data type:

C 'character value'

X 'hexadecimal value'

B 'binary value'

H '[+] decimal value'

The minimum value for H-type is -32768 and the maximum value is 32767.

F '[+] decimal value'

The minimum value for F-type is -2147483648 and the maximum is 2147483647.

E '[+] decimal value [E[+] decimal exponent]'

A maximum of 8 digits are allowed for the decimal value and a maximum of 2 digits are allowed for the decimal exponent.

D '[+] decimal value [E[+] decimal exponent]'

A maximum of 17 digits are allowed for the decimal value and a maximum of 2 digits are allowed for the decimal exponent.

P '[+] decimal value'

A maximum of 31 digits are allowed.

Z '[+] decimal value'

A maximum of 16 digits are allowed.

A '[+] decimal value'

The minimum decimal value is -2147483648 and the maximum value is 2147483647.

S 'decimal value(register number)'

The decimal value can be from 0 to 4095 and the register number must be from 0 to 15 (decimal form).

Y '[+] decimal value'

The decimal value may be from 0 to 32767.

You include your data following the code. Your data must be enclosed within apostrophes. Any single apostrophes within your data must be coded as two single apostrophes. Character data will be entered. 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 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.

The following restrictions apply to general and floating-point registers:

1. Only one data-type *should* be specified for floating-point registers. Additional values are ignored.

2. Assign only X or E data-types to single precision floating-point registers.

3. Assign only X or D data-types to double precision floating-point registers.

4. With the exception of the D-type of data, general registers can be assigned any data-type.

When a general register is assigned a list of data-type 'values', the first value is assigned to the specified register. Subsequent data-type values are assigned to contiguous higher-numbered registers. If register 15 is reached and data-type values remain, the values are wrapped around to register 0 and subsequent registers, if needed.

If data is assigned to a storage area that contains a breakpoint, the breakpoint is removed and a warning message is displayed at the terminal.

Example 1

Operation: Insert a character string at a particular location in virtual storage.

Known:

The address is a symbol: INPOINT
The data: January 1, 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'
```

AT Subcommand of TEST

Use the AT subcommand to establish breakpoints where processing is to be temporarily halted so that you can examine the results of execution up to the point of interruption. Processing is halted before the instruction at the breakpoint is executed.

MVS/XA: The AT subcommand sets breakpoints for all MVS/XA instructions except for the cross-memory instructions PC, PT, SAC, and SSAR.

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 prior to the execution of the breakpoint.
- A user written SVC exit.

```
AT          {address[:address]}
            {(address-list)   }
           [(subcommands-list)]
           [COUNT(integer)]
            [NODEFER]
            [DEFER  ]
            [NOTIFY  ]
            [NONOTIFY]
            [TITLE('text')]  16
```

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 an invalid instruction is encountered.

address-list
specifies several addresses that are to contain breakpoints. Each address must be on a halfword boundary. The list must be enclosed within parentheses, and the addresses in the list must be separated by standard delimiters (one or more blanks or a comma). A breakpoint is established at each address.

---

16 MVS/XA only

Note: For address, address:address, address-list, specify the 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-list
specifies one or more subcommands to be executed when the program is interrupted at the indicated location. If you specify more than one subcommand, the subcommands must be separated by semicolons. The list cannot be longer than 255 characters.

Note: If an OFF subcommand in the list removes the breakpoint for which a list is specified, all remaining subcommands in that list are ignored.

COUNT(integer)
specifies that processing 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 for 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. If both NOTIFY and NONOTIFY are omitted, NOTIFY is the default.

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

TITLE('text') MVS/XA Only
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.

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 (MVS/XA Only)

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.

ATTRIB Subcommand of TEST (MVS/XA Only)

Use the ATTRIB subcommand to build a list of attributes for non-VSAM data sets that are to be dynamically allocated. Refer to the ATTRIB command for a description of the syntax and function of the ATTRIB subcommand.

CALL Subcommand of TEST

Use the CALL subcommand to initiate processing at a specified address and to initialize registers 1, 14, and 15. You can pass parameters to the program that is to be tested.

Caution: The contents of registers 1, 14, and 15 are altered by the use of the CALL subcommand. To save the contents of these registers, use the COPY subcommand of TEST (see Examples 2 and 3 under the COPY subcommand).

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.

```
CALL          address
              [PARM(address-list)]
              [VL]
               [RETURN(address)]
               [RESUME]  17
              [AMODE     [(24)     ]]  17
                         [(31)     ]
                         [(SWITCH) ]
```

address
specifies the address where processing is to begin. Register 15 contains this address when the program under test begins execution.

You can specify the 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-list)
specifies one or more addresses that point to data to be used by the program being tested. The list of addresses are 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.

---

17 MVS/XA only

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 prior to the issuance of the CALL subcommand. If RETURN is omitted, register 14 contains the address of a breakpoint instruction.

RESUME (MVS/XA Only)
specifies upon completion of execution, the called program returns control to the address of the last breakpoint prior to the CALL.

AMODE [**(24)** / **(31)** / **(SWITCH)**] **MVS/XA Only**
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.

Example 1

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

Known:

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

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

Note: The following message is issued after completion of the called routine:

```
'IKJ57023I PROGRAM UNDER TEST HAS TERMINATED NORMALLY+'
```

This message is issued because no return address was specified. If GO is now specified without an address, the TEST session 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)
```

CANCEL Subcommand of TEST (MVS/XA Only)

Use the CANCEL subcommand to halt processing of batch jobs submitted from the terminal. Refer to the CANCEL command for a description of the syntax and function of the CANCEL subcommand.

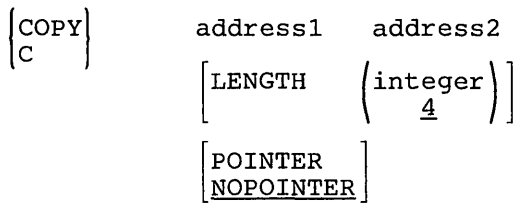
COPY Subcommand of TEST

Use the COPY subcommand to transfer data or addresses from:

- One storage address to another
- One general register to another
- A register to virtual storage
- Virtual storage to a register.

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.



---

address1
specifies a location that contains data to be copied.

address2
specifies a location that receives the data after it is copied.

You can specify address1 and address2 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.

LENGTH(integer) or 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 address1 is to be validity checked to see that it does not exceed maximum virtual storage size. Address1 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 address2. When using the POINTER operand, do not specify a general register as address1.

NOPOINTER
specifies address1 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 or 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 address1, the maximum length of data transferred is the total length of the general registers or 64 bytes.

When the value of address2 is one greater than address1, propagation of the data in address1 occurs. When the value of address2 is more than one greater than the value of address1, no propagation occurs.

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. 1(64)
```

Example 4

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

Known:

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

```
c 80680. 80681. 1(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 1(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. 1(8) nopoint
```

DELETE Subcommand of TEST

Use the DELETE subcommand to delete, from virtual storage, a load module that was loaded by the tested program, or one of its subtasks.

MVS/XA: 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.

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

---

load-module-name
specifies the name of the load module to be deleted. The load name is the name (which might be an alias) by which the program is known to the system when it is in virtual storage. The name must not exceed eight characters.

Example 1

Operation: Delete a load module from virtual storage.

Known:

The name of the load module: TOTAL

```
delete total
```

or

```
del total
```

DROP Subcommand of TEST

Use the DROP subcommand to remove symbols from the symbol table of the module being tested. You can only remove symbols that you established with the EQUATE subcommand or 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.

```
DROP        (symbol-list)
```

(symbol-list)
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.

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)
```

END Subcommand of TEST

Use the END subcommand to terminate all functions of the TEST command and the program being tested.

```
END
```

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.

EQUATE Subcommand of TEST

Use the EQUATE subcommand to add a symbol to the symbol table of the module being tested. This subcommand allows you to establish a new symbol that you can use to refer to an address or 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. For restrictions on symbols see the section titled, "Internal Symbols." You can also modify the data attributes (type, length, and multiplicity); use the EQUATE subcommand to modify attributes of existing equated symbols. The DROP subcommand removes symbols added by the EQUATE subcommand. Symbols established by the EQUATE subcommand are defined for the duration of the TEST session only.

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

---

symbol

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

address

specifies the address is to equate to the symbol that you specified.

You can specify the 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
Z	Zoned decimal	16
A	Address constant	4
S	Address (base + displacement)	2
Y	Address constant (halfword)	2

LENGTH(integer)

specifies the length of the data. The maximum value of the integer is 256. If you do not specify the length, the following default values 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.

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
```

EXEC Subcommand of TEST (MVS/XA Only)

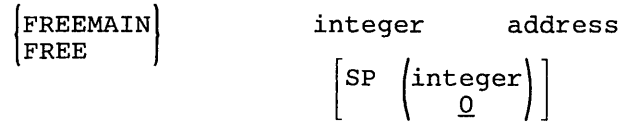
Use the EXEC subcommand to execute a CLIST. Refer to the EXEC command for a description of the syntax and function of the EXEC subcommand.

Only TEST subcommands and CLIST statements should be specified in the CLIST. You can enter any TSO command in the CLIST after entering END or RUN to terminate TEST.

FREEMAIN Subcommand of TEST

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

MVS/XA: Use the FREEMAIN subcommand to free a specified number of bytes of virtual storage above or below 16Mb.



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.

The LISTMAP subcommand can be used to help locate previously acquired virtual storage.

You can specify the 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) or 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.

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)
```

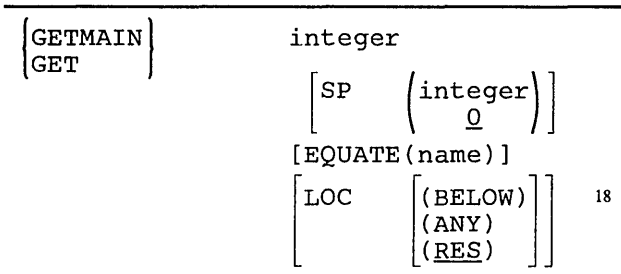
Warning: Do not attempt to free all of subpool 78. If you want to free a portion of subpool 78, be careful not to free the storage obtained by the TMP. This results in freeing the TMP's data areas because subpool 78 is shared. The deletion of the TMP portion of subpool 78 causes your session to terminate.

You can release an entire subpool by specifying a length of 0, an absolute address of 0, and a subpool in the range of 1 through 127.

If you specify a non-zero address, the length must also be non-zero.

GETMAIN Subcommand of TEST

Use the GETMAIN subcommand to obtain a specified number of bytes of virtual storage. The GETMAIN subcommand displays the starting address of the virtual storage obtained.



integer
specifies the number of bytes, in decimal form, of virtual storage to be obtained.

SP(integer) or 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) MVS/XA Only
specifies the virtual and real storage area must be below 16 Mb.

LOC (ANY) MVS/XA Only
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) MVS/XA Only
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 16Mb, the request is processed as LOC (ANY). LOC (RES) is the default.

---

18 MVS/XA only

Example 1

Operation: Obtain 240 decimal bytes of virtual storage from subpool 0.

```
getmain 240
```

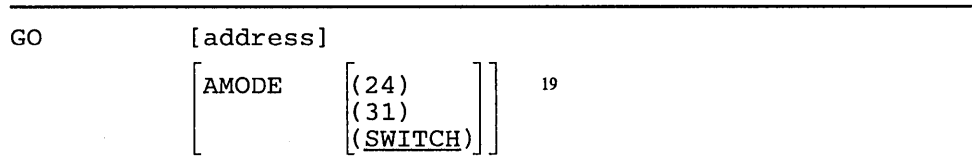
Example 2

Operation: Obtain 500 bytes of virtual storage from subpool 3 and equate starting address to symbolic name AREA.

```
get 500 sp(3) equate(area)
```

GO Subcommand of TEST

Use the GO subcommand to start or restart program execution from a particular address. If the program was interrupted for a breakpoint and you want to continue from the breakpoint, there is no need to specify the address. However, you can start execution at any point by specifying the address.



address
specifies a symbolic address, a relative address, an absolute address, or a general register containing an address. Execution begins at the address that you specify.

When the problem program completes processing, the following message is displayed at the terminal:

```
'IKJ57023I PROGRAM UNDER TEST HAS TERMINATED NORMALLY+'
```

If the GO subcommand is then issued with no address specified, the TEST session is terminated.

AMODE [**(24)** | **(31)** | **(SWITCH)**] **MVS/XA Only**
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.

19 MVS/XA only

Example 1

Operation: Begin execution of a program at the point where the last interruption occurred or initiate execution of a program.

```
go
```

Example 2

Operation: Begin execution at a particular address.

```
go calculat
```

HELP Subcommand of TEST

Use the HELP subcommand to obtain the syntax and function of the TEST subcommands. Refer to the HELP command for a description of the syntax and function of the HELP subcommand.

LINK Subcommand of TEST (MVS/XA Only)

Use the LINK subcommand to invoke the linkage editor service program. Refer to the LINK command for a description of the syntax and function of the LINK subcommand.

LIST Subcommand of TEST

Use the LIST subcommand to have the contents of a specified area of virtual storage or the contents of registers displayed at your terminal or placed into a data set.

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

address
specifies the location of data that you want displayed at your terminal or placed into a data set.

address:address
specifies that you want the data located between the specified addresses displayed at your terminal or placed into a data set.

(address-list)
specifies several addresses of data that you want displayed at your terminal or placed into a data set. The data at each location 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 or general). The list of addresses must be enclosed within parentheses, and the addresses must be separated by standard delimiters (one or more blanks or a comma).

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.

You can specify address, address:address, and address-list 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.

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
Z	Zoned decimal	16
A	Address constant	4
S	Address (base + displacement)	2
Y	Address constant (halfword)	2

All accepted data-types allow the specified address to be aligned on a byte boundary even though certain data-types cannot be assigned to a byte boundary. The default for data-type is hexadecimal.

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.

If an area is to be displayed using the I data-type and the area contains an invalid op code, only the area up to that invalid op code is displayed.

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

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 7).
- A way of printing more than 256 bytes at a time. The value supplied for integer determines how many lengths or multiples of data-type you want listed. The value supplied for integer cannot exceed 256.

For I type data, the value supplied for MULTIPLE defines the number of instructions to be displayed. If you use a symbolic address and do not specify either LENGTH or MULTIPLE, the length retrieved from the internal TEST symbol table or from the program 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 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 quotes, the descriptive qualifier TESTLIST is added.

If your record type was:	**Fixed, Fixed Blocked, or Undefined**		**Variable or Variable Blocked**	
Then it is changed to variable blocked with the following attributes:	Recordsize	Blocksize	Recordsize	Blocksize
	125	1629	125	129

Record and block sizes greater than those specified in the preceding table are unchanged.

The specified data set is kept open until:

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

Example 1

Operation: List the contents of floating-point register 2 in single precision.

```
list  2e
```

Example 2

Operation: List all of the general registers.

```
list  0r:15r
```

Example 3

Operation: List all of the floating point registers in double precision.

```
list  0d:6d
```

Example 4

Operation: List 20 instructions starting with address +3A

```
list +3a i m(20)
```

Example 5

Operation: List the contents of an area of virtual storage.

Known:

The area to be displayed is between labels COUNTERA and DTABLE.

The data is to be listed in character format for a length of 130 bytes.

The name of the data set 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 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 7

Operation: List the first six fullwords in the communications vector table (CVT).

Known:

The absolute address of the CVT: 10

The user is operating in TEST mode.

The data is to be listed in hexadecimal form in six lines of 4 bytes each.

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

```
qualify 10.%
```

TEST: The system response

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

The display at your terminal will resemble the following:

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

In the preceding example, the hexadecimal data-type was not specified. It was the default.

LISTALC Subcommand of TEST (MVS/XA Only)

Use the LISTALC subcommand to obtain a list of names of the data sets allocated during the current user session. Refer to the LISTALC command for a description of the syntax and function of the LISTALC subcommand.

LISTBC Subcommand of TEST (MVS/XA Only)

Use the LISTBC subcommand to obtain a listing of the contents of the broadcast data set, SYS1.BRODCAST. It contains messages of general interest (NOTICES) and messages directed to particular users (MAIL). Refer to the LISTBC command for a description of the syntax and function of the LISTBC subcommand.

LISTCAT Subcommand of TEST (MVS/XA Only)

Use the LISTCAT subcommand to list catalog entries by name of entry type and selected fields for each entry. Refer to the LISTCAT command for a description of the syntax and function of the LISTCAT subcommand.

LISTDCB Subcommand of TEST

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

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.

```
LISTDCB       address
              [FIELD(names)]
              [PRINT(data-set-name)]
```

address
specifies the address of the DCB that you want displayed. The address must be on a fullword boundary.

You can specify the 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(names)
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.

Following is a list of the valid field names for the DCB:

DCBBFALN	DCBIFLGS
DCBBFTEK	DCBIOBAD
DCBBUFCB	DCBKEYCN
DCBBUFL	DCBKEYLE
DCBBUFNO	DCBMACRF
DCBDDNAM	DCBOFLGS
DCBDEBAD	DCBRECFM
DCBDEVT	DCBRELAD
DCBDVTBL	DCBTIOT
DCBEODAD	DCBTRBAL
DCBEXLST	DCBMACR
DCBFDAD	DCBDSORG
DCBHIARC	

PRINT(data-set-name)

specifies the name of the sequential data set to which data is to be directed. If you omit this operand, the data is displayed at your terminal.

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

If the data-set-name is not specified within quotes, the descriptive qualifier TESTLIST is added.

If PRINT(data-set-name) is specified, use the following table to determine the format of the output.

If your record type was:	**Fixed, Fixed Blocked, or Undefined**		**Variable or Variable Blocked**	
Then it is changed to variable blocked with the following attributes:	Recordsize	Blocksize	Recordsize	Blocksize
	125	1629	125	129

Record and block sizes greater than those specified in the preceding table are unchanged.

The specified data set is kept open until:

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

Example 1

Operation: List the RECFM field of a DCB for the program that is being tested.

Known:

The DCB begins at location: DCBIN

```
listdcb dcbin field(dcbrecfm)
```

Example 2

Operation: List an entire DCB.

Known:

The absolute address of the DCB: A33B4

```
listdcb a33b4.
```

LISTDEB Subcommand of TEST

Use the LISTDEB subcommand to list the contents of a data extent block (DEB). You must provide the address of the DEB.

MVS/XA: 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.

```
LISTDEB          address
                 [FIELD(names)]
                 [PRINT(data-set-name)]
```

address
specifies the address is the beginning of the DEB. It must be on a fullword boundary.

You can specify the 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(names)
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.

Following is a list of DEB names that are valid for the LISTDEB subcommand:

DEBAMLNG	DEBNMEXT	DEBUSRPG
DEBAPPAD	DEBNMSUB	
DEBDCBAD	DEBOFLGS	
DEBDEBAD	DEBOPATB	
DEBDEBID	DEBPRIOR	
DEBECBAD	DEBPROTG	
DEBEXSCL	DEBQSCNT	
DEBFLGS1	DEBTCBAD	
DEBIRBAD	DEBUSPRG	

Following is a list of the valid DEB names in the direct access section:

DEBBINUM	DEBNMTRK
DEBDVMOD	DEBSTRCC
DEBENDCC	DEBSTRHH
DEBENDHH	DEBUCBAD

Note: These fields cannot be accessed unless there is a direct access section in the DEB.

PRINT(data-set-name)

specifies the name of the sequential data set to which data is to be directed. If you omit this operand, the data is displayed at your terminal.

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

If the data-set-name is not specified within quotes, the descriptive qualifier TESTLIST is added.

If PRINT(data-set-name) is specified, use the following table to determine the format of the output.

If your record type was:	**Fixed, Fixed Blocked, or Undefined**		**Variable or Variable Blocked**	
Then it is changed to variable blocked with the following attributes:	Recordsize	Blocksize	Recordsize	Blocksize
	125	1629	125	129

Record and block sizes greater than those specified in the preceding table are unchanged.

The specified data set is kept open until:

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

Example 1

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

Known:

The address of the DEB is 44 decimal (2C hexadecimal) bytes past the beginning of the DCB.

The address of the DEB: DCBIN+2C%

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

Example 2

Operation: List the following fields in the DEB: DEBDCBAD and DEBOFLGS

Known:

The address of the DEB is 44 decimal (2C hexadecimal) bytes past the beginning of the DCB. The address of the DCB is in register 8.

```
listdeb 8r%+2c% field(debdcbad,deboflgs)
```

LISTDS Subcommand of TEST (MVS/XA Only)

Use the LISTDS subcommand to display attributes of specific data sets at the terminal. Refer to the LISTDS command for a description of the syntax and function of the LISTDS subcommand.

LISTMAP Subcommand of TEST

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

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

```
LISTMAP          [PRINT(data-set-name)]
```

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

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

If the data-set-name is not specified within quotes, the descriptive qualifier TESTLIST is added.

If PRINT(data-set-name) is specified, use the following table to determine the format of the output.

If your record type was:	**Fixed, Fixed Blocked, or Undefined**		**Variable or Variable Blocked**	
Then it is changed to variable blocked with the following attributes:	Recordsize	Blocksize	Recordsize	Blocksize
	125	1629	125	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 subcommand is entered that specifies a different PRINT data set. In this case, the former data set is closed and the current one opened.

Example 1

Operation: Display a map of virtual storage at your terminal.

```
listmap
```

Example 2

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

Known:

The name of the data set: ACDQP.MAP.TESTLIST
The prefix in the user's profile: ACDQD

```
listmap print(map)
```

LISTPSW Subcommand of TEST

Use the LISTPSW subcommand to display a program status word (PSW) at your terminal.

```
LISTPSW          [ADDR(address)]
                 [PRINT(data-set-name)]
```

ADDR(address)
specifies the address which identifies a particular PSW. If you do not specify an address, you receive the current PSW for the program that is executing.

You can specify the 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 the sequential data set to which data is to be directed. If you omit this operand, the data is displayed at your terminal.

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

If the data-set-name is not specified within quotes, the descriptive qualifier TESTLIST is added.

If PRINT(data-set-name) is specified, use the following table to determine the format of the output.

If your record type was:	**Fixed, Fixed Blocked, or Undefined**		**Variable or Variable Blocked**	
Then it is changed to variable blocked with the following attributes:	Recordsize	Blocksize	Recordsize	Blocksize
	125	1629	125	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 subcommand is entered that specifies a different PRINT data set. In this case, the former data set is closed and the current one is opened.

Example 1

Operation: Display the current PSW at your terminal.

```
listpsw
```

Example 2

Operation: Direct the input/output old PSW into a data set.

Known:

The prefix in the user's profile: ANZAL2
The address of the PSW (in hexadecimal): 38
The name of the data set: ANZAL2.PSWS.TESTLIST

```
listpsw addr(38.) print(psws)
```

LISTTCB Subcommand of TEST

Use the LISTTCB subcommand to display the contents of a task control block (TCB). You can provide the address of the beginning of the TCB.

MVS/XA: If a copy of the control block is in extended virtual storage, the LISTTCB subcommand accepts addresses greater than 16 Mb, even though the block itself is below 16 Mb 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.

```
LISTTCB          [ADDR(address)]
                 [FIELD(names)]
                 [PRINT(data-set-name)]
```

ADDR(address)
specifies the address must be on a fullword boundary. The address identifies the particular TCB that you want to display. If you omit an address, the TCB for the current task is displayed.

You can specify the 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(names)
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.

MVS/370 and MVS/XA: Following is a list of the valid LISTTCB field names.

TCBABCUR	TCBIOBRC	TCBRBP
TCBAECB	TCBIQE	TCBRCMP
TCBAFFN	TCBJLB	TCBRTM12
TCBAQE	TCBJPQ	TCBRTWA
TCBBACK	TCBJSCB	TCBSTABB
TCBCCPVI	TCBJSTCB	TCBSTMCT
TCBCMP	TCBLLS	TCBSTPCT
TCBDAR	TCBLMP	TCBSWA
TCBDEB	TCBLCT	TCBSYSCT
TCBDSP	TCBMSS	TCBTCB
TCBECB	TCBNDSPO	TCBTCBID
TCBESTAE	TCBNDSP1	TCBTCT
TCBEXT1	TCBNDSP2	TCBTFLG
TCBEXT2	TCBNDSP3	TCBTID
TCBBYT1	TCBNDSP4	TCBTIO
TCBFLGS	TCBNDSP5	TCBTME
TCBFLGS6	TCBNSTAE	TCBTRN
TCBFLGS7	TCBNTC	TCBTSDP
TCBFOE	TCBOTC	TCBSTFLG
TCBFSAB	TCBPIE	TCBTSLP
TCBGRS	TCBPKE	TCBUSER
TCBGTFA		

MVS/370: Following is a list of the valid LISTTCB field names.

TCBDDEXC	TCBIOTIM	TCBQEL
TCBDDWTC	TCBPQE	TCBTIRB
		TCBMSAV

MVS/XA: Following is a list of the valid LISTTCB field names.

TCBAE	TCBNEEP	TCBSSAT
TCBCANF	TCBPERCP	TCBSTAWA
TCBEAE	TCBPERCT	TCBTQE
TCBERD	TCBRD	TCBXLAS
TCBEVENT	TCBSEQNO	TCBXSB
		TCBXSCT1

PRINT(data-set-name)

specifies the name of the sequential data set to which data is to be directed. If you omit this operand, the data is be displayed at your terminal.

The data format is variable-length blocked records. Old data sets with the standard record 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 data-set-name is not specified within quotes, the descriptive qualifier TESTLIST is added.

If PRINT (data-set-name) is specified, use the following table to determine the format of the output.

If your record type was:	**Fixed, Fixed Blocked, or Undefined**		**Variable or Variable Blocked**	
Then it is changed to variable blocked with the following attributes:	Recordsize	Blocksize	Recordsize	Blocksize
	125	1629	125	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 a END subcommand, or
- A LIST subcommand is entered that specifies a different PRINT data set. In this case, the former data set is closed and the current one opened.

Example 1

Operation: Direct a copy of the TCB for the current task into a data set.

Known:

The prefix in the user's profile is NAN75.
The name of the data set: NAN75.TCBS.TESTLIST

```
listtcb print(tcbs)
```

Example 2

Operation: Save a copy of some fields of a task's control block that is not active in a data set for future information.

Known:

The symbolic address of the TCB: MYTCB2
The fields that are being requested: TCBTIO TCBCMP TCBGRS
The name of the data set: SCOTT.TCBDATA

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

Example 3

Operation: List the entire TCB for the current task.

```
listtcb
```

LOAD Subcommand of TEST

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

MVS/XA: Use the LOAD subcommand to load a program above or below 16 Mb 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 in order to obtain the actual address.

```
LOAD          data-set-name
```

data-set-name
specifies the name 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 quotes, the LOAD qualifier is added.

Example 1

Operation: Load a program named GSCORES from the data set ATX03.LOAD.

Known:

The prefix in the user's profile is ATX03.

```
load 'atx03.load (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:

```
test.load(attempt)
```

Example 3

Operation: Load a module named PERFORM from data set ATX03.TRY.

```
load 'atx03.try(perform)'
```

OFF Subcommand of TEST

Use the OFF subcommand to remove breakpoints from a program.

```
OFF        [address[:address]]
           [(address-list)    ]
```

address
specifies the location of a breakpoint that you want to remove. The address must be on a halfword boundary.

If no address is specified, all breakpoints are removed. You can specify the 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-list)
specifies the location of several breakpoints that you want to remove. See the description of address for a list of valid address types.

Note: The list *must* be in parentheses with address separated by one or more blanks or a comma.

Example 1

Operation: Remove all breakpoints in a section of 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%
```

OR Subcommand of TEST (MVS/XA Only)

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 register to virtual storage
- Virtual storage to a register.

```
OR          address1   address2
           [LENGTH    (integer)]
                      (  4   )
           [POINTER  ]
           [NOPOINTER]
```

---

address1
specifies the location of data that is to be ORed with data pointed to by address2.

If you do not specify POINTER and there is a breakpoint in the data pointed to by address1, the TSO TEST processor terminates the OR operation.

address2
specifies the location of the data that is to be ORed with data pointed to by address1. When the OR operation is complete, the result is stored at this location.

You can specify address1 and address2 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).

LENGTH(integer) or 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 address1 is to be validity checked to see that it does not exceed maximum virtual storage size. Address1 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 address1.

NOPOINTER
specifies address1 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 address1, the maximum length of data that is ORed is the total length of the general registers or 64 bytes.

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 1(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. 1(8)
```

PROFILE Subcommand of TEST (MVS/XA Only)

Use the PROFILE subcommand to establish, change, or list your user profile. Refer to the PROFILE command for a description of the syntax and function of the PROFILE subcommand.

PROTECT Subcommand of TEST (MVS/XA Only)

Use the PROTECT subcommand to prevent unauthorized access to a non-VSAM data set. Refer to the PROTECT command for a description of the syntax and function of the PROTECT subcommand.

QUALIFY Subcommand of TEST

Use the QUALIFY subcommand to qualify symbolic and relative addresses; that is, to establish the starting or base location to which displacements are added so that an absolute address is obtained. The QUALIFY subcommand allows you to 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. See the subsection titled "Qualified Addresses" at the beginning of this section for a more detailed description of qualified addresses.

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

```
┌QUALIFY┐   ┌address                                          ┐
└Q      ┘   └module-name[.entryname]  [TCB(address)]         ┘
```

address
specifies the base location to be used in determining the absolute address for relative addresses only. It does not affect symbolic addressing. You can specify the 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[.entryname[
specifies the name by which a load module is known, and optionally, an externally referable name within a module. If only a module is specified, the first entry point in the module will be supplied.

TCB(address)
specifies the address of a task control block (TCB). This operand is necessary when programs of the same name are assigned to two or more subtasks and you must establish uniquely which one is to be qualified.

Note: When using 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.

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.

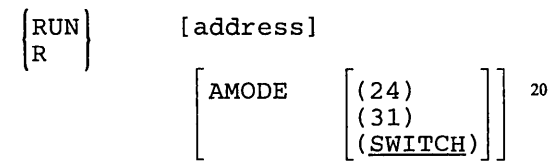
```
q salesrpt tcb(5r%)
```

RENAME Subcommand of TEST (MVS/XA Only)

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

RUN Subcommand of TEST

Use the RUN subcommand to cause the program that is being tested to execute to termination without recognizing any breakpoints. When you specify this subcommand, TEST is terminated. When the program completes, you can enter another command. Overlay programs are not supported by the RUN subcommand. Use the GO subcommand to execute overlay programs.



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 the 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)**] **MVS/XA Only**

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.

---

20 MVS/XA only

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 abends 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.

Example 1

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

```
run
```

Example 2

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

Known:

The address: +A8

```
run +a8
```

SEND Subcommand of TEST (MVS/XA Only)

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

STATUS Subcommand of TEST (MVS/XA Only)

Use the STATUS subcommand to display the status of batch jobs at the terminal. Refer to the STATUS command for a description of the syntax and function of the STATUS subcommand.

SUBMIT Subcommand of TEST (MVS/XA Only)

Use the SUBMIT subcommand to submit one or more batch jobs for processing under TEST. Refer to the SUBMIT command for a description of the syntax and function of the SUBMIT subcommand.

TERMINAL Subcommand of TEST (MVS/XA Only)

Use the TERMINAL subcommand to define the operating characteristics for the type of terminal being used. Refer to the TERMINAL command for a description of the syntax and function of the TERMINAL subcommand.

UNALLOC Subcommand of TEST (MVS/XA Only)

Use the UNALLOC subcommand to release (deallocate) previously allocated data sets which are no longer needed. UNALLOC is issued instead of FREE under TEST. The syntax and operands are identical.

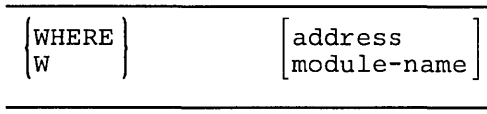
WHERE Subcommand of TEST

Use the WHERE subcommand to obtain:

- An absolute address
- The name of a module and CSECT
- A relative offset within the CSECT
- 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.



address

You can specify the 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.

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)
- Service units used, which can be:

  CPU Service Units - The task execution time, divided by an SRM constant, that is CPU model-dependent.

  I/O Service Units - The sum of individual SMF data set activity EXCP counts for all data sets associated with the address space.

  Storage Service Units - The number of real page frames multiplied by CPU service units, multiplied by .02. The decimal .02 is a scaling factor designed to bring the storage service component in line with the CPU component.

- Local time of day

  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 upon the executing program.

TIME

TRANSMIT Command

The TRANSMIT command allows you to send information (a message), or a copy of information (a data set), or both, from one user to another. 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.

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 can transmit to VM is 65,535. Data sets with machine and ASA print-control characters are also supported. TRANSMIT does not support data sets with keys, ISAM data sets, VSAM data sets, or data sets with user labels.

If a partitioned data set (PDS) is transmitted, it is unloaded with IEBCOPY and then the unloaded version is transmitted. If a single member of a PDS is transmitted, it is generally unloaded before transmission. You can force transmission of a PDS as a sequential data set by using the SEQUENTIAL operand. Forced transmission of a PDS 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 PDS.

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 messages are shown before you are prompted to indicate what to do with the data. You can enter messages in either full screen mode or single line mode.

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 is 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.

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.

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.

Data Encryption Function of TRANSMIT and RECEIVE

The TRANSMIT and RECEIVE commands support encryption using the following program products:

Access Method Services Cryptographic Option and either Programmed Cryptographic Facility or Cryptographic Unit Support (**MVS/370**)

Data Facility Product (DFP) (**MVS/XA**)

TSO 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 program products 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.

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. In the absence of any user or installation specification, the default log data set name is 'prefix.LOG.MISC'.

2. 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.

3. 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 - 79	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 - 79	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 - 79	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 'userid.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 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 below, it is treated as a user-defined tag that may be processed by a user application. The information that follows a user-defined tag is ignored by TRANSMIT and RECEIVE processing.

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. names-file-dataset-name

:EPILOG. epilog line

:PROLOG. prolog line

:LOGNAME. log-dataset-last-qualifier

:LOGSEL. log-dataset-middle-qualifier

:LOG or :NOLOG (The default is :LOG.)

:NOTIFY or :NONOTIFY (The default is :NOTIFY.)

Tag Definitions

: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 the 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 should 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 file. Text data for the :PROLOG tag should 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."

:LOGSEL.name
in the control section, specifies the second (middle) qualifier(s) of all log data sets. See "Logging Function of TRANSMIT and RECEIVE."

:LOG or :NOLOG
in the control section, indicates whether or not 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 prompts you with a message to receive data set 'A.MAIL.USERID'. TSO 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.

:NOTIFY or :NONOTIFY
in the control section, indicates whether or not 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).

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/or :CC tags 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 following tags as all uppercase or all lowercase.

   :NOTIFY or :NONOTIFY

   :NICK. nickname (Required)

   :NODE. nodename (Default is your own node.)

   :USERID. user ID (Required)

   :LOG/:NOLOG

   :LOGLST or :NOLOGLST

   :NAME. username

   :ADDR.address

   :LIST.name name-list

   :CC.name name-list

   :PARM.text

Tag Definitions

:NOTIFY or :NONOTIFY
in the control section, specifies whether or not 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 one-character to eight-character string of non-blank alphanumeric characters.

:NODE.nodeid
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.userid
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 or :NOLOG
in the control section, indicates whether or not 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 prompts you with a message to receive data set 'A.MAIL.USERID'. TSO 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 or not a log entry should be made for each addressee in the list.

:NAME.username
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 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.

:PARMT.text
specifies up to 30 characters of installation-defined data. TSO passes this data to the RECEIVE command installation exits. See *System Programming Library: User Exits and Modifications* for more information about how an installation uses these exits.

The format of the TSO TRANSMIT command is:

```
{TRANSMIT}           (addressee-list)
{XMIT    }      [{   {DATASET(dsn)   }    [DSNAME(dsn)   ]}]
                [{   {DDNAME(ddname) }    [FILE(ddname)  ]}]
                [{TERMINAL                                }]
                 [MESSAGE]
                 [MSG    ]
                 [COPYLIST  ]
                 [NOCOPYLIST]
                 [ENCIPHER]
                 [EPILOG  ]
                 [NOEPILOG]
                 [FULLSCREEN]
                 [LINE      ]
                 [LINE(nn)  ]
                 [LOG     ]
                 [NOLOG   ]
                 [LOG(ALL)]
                 [LOGNAME(name)]
                 [MEMBERS(memberlist)]
                 [NOTIFY     ]
                 [NOTIFY(ALL)]
                 [NONOTIFY   ]
                  [PARM(parameters)]
                 [PDS       ]
                 [SEQUENTIAL]
                 [PROLOG  ]
                 [NOPROLOG]
                 [SYSOUT(sysoutclass or *)]
                [{OUTDDNAME(ddname)}]
                [{OUTFILE(ddname)  }]
                [{OUTDSNAME(dsn)   }]
                [{OUTDATASET(dsn)  }]
```

(addressee-list)
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.userid or node/userid, 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."

DATASET(dsn) or DSNAME(dsn)
specifies the name of a data set to be transmitted. The data set must be on a direct access storage device (DASD).

DDNAME(ddname) or FILE(ddsname)
specifies the 1 to 8 character DD name of a preallocated file to be transmitted. The data set must be on a direct access storage device (DASD). 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.

MESSAGE or 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 prompts you twice for the data.
- TSO uses the prefix as the high level qualifier for the name of the data set to be transmitted.

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
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
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 or LINE(nn)
requests terminal input for messages and data be read in single line mode. This is the default for non-3270 terminals. 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 one or two 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
records the transmission in the LOG data set. LOG does not necessarily indicate that the log entry will contain 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 will contain 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 unless NOLOG is specified. See "Logging Function of TRANSMIT and RECEIVE."

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.

LOGNAME(name)
uses the name as the LOGNAME qualifier on the log data set name. See "Logging Function of TRANSMIT and RECEIVE."

MEMBERS(memberlist)
transmits a list of members from the specified partitioned data set.

NOTITY
notifies the sender when the data has been received. NOTIFY does not necessarily guarantee that notification will be 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 nicknames 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(parameters)
Your installation may instruct you to use this operand to specify installation dependent data.

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.

SEQUENTIAL
sends a member of a PDS 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 PDS.

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 or *)
uses the SYSOUT class for messages from utility programs, which are used by TRANSMIT (for example IEBCOPY). If you specify a * (asterisk), TSO directs utility program messages to the terminal. The default is usually *, but the installation can modify it.

OUTDDNAME(ddname) or 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. TSO assigns the DCB attributes as LRECL=80, BLKSIZE=3120, and RECFM=FB. Specify the DD name as either a sequential data set or a member of a partitioned data set.

Use OUTDDNAME or OUTFILE in conjunction with the INDDNAME or INFILE operand of the RECEIVE command. OUTDDNAME and OUTFILE are primarily intended for system programmer use.

OUTDSNAME(dsn) or OUTDATASET(dsn)
specifies the use of a data set as the output data set for the TRANSMIT command. No data is written to SYSOUT for transmission. TSO assigns the DCB attributes as LRECL=80, BLKSIZE=3120, and RECFM=FB. The DD name must be a sequential data set.

Use OUTDSNAME or OUTDATASET in conjunction with the INDSNAME or INDATASET operand of the RECEIVE command. OUTDSNAME and OUTDATASET are primarily intended for system programmer use.

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 lower case and data displayed by the system is in upper case.

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 ALLOCATION PARAMETERS, 'DELETE', OR 'END'
<null line>
RESTORE SUCCESSFUL TO DATASET 'USER2.PARMLIB'
---------------------------------------------
No more Interactive Data Transmission Facility files
are available for the RECEIVE command to process.
```

In the above 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,ieaips00) msg 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.
<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 aborts 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
MEMBERS: IEASYS00, IEAIPS00
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 ALLOCATION PARAMETERS, 'DELETE', OR 'END'
end
```

In the above 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 upper and lower case 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 in order to leave space for later additions.

```
receive
DATASET 'SYS1.PARMLIB'       FROM USER1 ON NODEA
MEMBERS: IEASYS00, IEAIPS00
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 ALLOCATION PARAMETERS, 'DELETE', OR 'END'
da('nodea.parmlib') space(1) cyl dir(10)
RESTORE SUCCESSFUL TO DATASET 'NODEA.PARMLIB'
------------------------------------------
No more Interactive Data Transmission Facility files
are available for the RECEIVE command to process.
```

The received members IEASYS00 and IEAIPS00 are 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 following text is entered on successive lines>
<of a full-screen data area.                       >
Davy,
   I sure would like to have a coonskin cap like
yours.
<Use PF3 to cause message to be sent>
```

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. A full-screen input area is displayed to the user. In this area, he can type his data, scroll using program function (PF) keys PF7 or PF19 and PF8 or PF20, and exit using PF3 or PF15.

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.

```
TSOEXEC  [TSO command]
```

[TSO command]
specifies any TSO command the TSO Service Facility can invoke, whether or not the command is authorized or unauthorized.

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')
```

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          SYSRC(operator  integer)
             [END          ]
             [command-name ]
```

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 command name and appropriate operands. If the comparison is true, TSO processes the command.

WHEN terminates CLIST processing and then executes the TSO command name specified.

Use successive WHEN commands to determine an exact return code and then perform some action based on that return code.

Example

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
```

The CLIST Statements

This section contains descriptions of the CLIST statements. They are presented in alphabetical order. Note that *CLISTs: Implementation and Reference* is the authoritative source for information about CLISTs such as the following:

- Operators and expressions
- Symbolic variables
- Control variables
- Built-in functions
- Error codes.

ATTN CLIST Statement

Use the ATTN statement to set up an environment that detects attention interruptions processed by the terminal monitor program (TMP). The detection of an attention interruption invokes a specified action which is considered to be an attention exit.

For more information on this statement and the writing of CLISTs, see *CLISTs: Implementation and Reference.*

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

---

label:
specifies a name to which the CLIST can branch. Enter one-to-eight characters, the first alphabetic and the rest alphameric, followed by a colon.

OFF
specifies any previous attention action is nullified. When no action is specified on the ATTN statement, OFF is the default.

action
Specifies either:

1. One TSO command (commonly an EXEC of another CLIST) or a null.

2. A DO-group constituting an attention exit routine. The exit would consist of either one TSO command (commonly an EXEC of another CLIST), a null statement, or an EXIT statement.

   Results:

   Null--The attention is ignored.
   TSO command--Control is given to the specified command.
   EXIT statement--The attention is ignored, and the CLIST is terminated.

If any statements occur after the one TSO command, a null statement, or an EXIT statement, they are ignored.

In order to prevent the main CLIST from being deleted by a stack flush request from the system after the ATTN action has been processed, you must specify CONTROL MAIN in the CLIST. CONTROL NOFLUSH does not prevent the stack from being flushed during an ATTN action.

This is also the case if you use GLOBAL variables in the main CLIST and in a CLIST that is invoked by the ATTN action. To maintain GLOBAL variables, specify GLOBAL MAIN in the CLIST. CONTROL NOFLUSH does not prevent the GLOBAL variables from being flushed during an ATTN action.

Example

Operation: Pass control to a CLIST from an attention exit.

```
.
.
ATTN     DO
         WRITE DO YOU WANT TO TERMINATE (Y OR N)
         READ &ANS
         IF &ANS=Y THEN +        /* IF TERMINATION REQUESTED    */
            EXEC (CLEANUP)       /* -THEN CLEANUP AND END       */
         ELSE                    /* IF NO TERMINATION REQUESTED */
            DO                   /* -THEN SPECIFY A NULL        */
                 SET &CMD=       /* SET VARIABLE TO NULL        */
                 &CMD            /* SPECIFY NULL TO CONTINUE    */
             END
         END
ALLOC F(INPUT) DA(INPUT.TEXT)
ALLOC F(OUTPUT) DA(OUTPUT.TEXT)
ALLOC F(TEMP) DA(TEMP.TEXT)
CALL 'MYID.MYPROG.LOAD(MEMBER)'
FREE F(INPUT) DA(INPUT.TEXT)
FREE F(OUTPUT) DA(OUTPUT.TEXT)
FREE F(TEMP) DA(TEMP.TEXT)
END
```

The CLIST CLEANUP contains:

```
/*THIS CLIST IS INVOKED WHEN TERMINATION IS REQUESTED FROM     */
/*THE ATTENTION ROUTINE. IT WILL FREE ALL OF THE CURRENTLY     */
/*ALLOCATED FILES.
FREE F(INPUT) DA(INPUT.TEXT)
FREE F(OUTPUT) DA(OUTPUT.TEXT)
FREE F(TEMP) DA(TEMP.TEXT)
```

CLOSFILE CLIST Statement

Use the CLOSFILE statement to close a file that was previously opened by an OPENFILE statement. It is not necessary to specify file type. You can close only one file with one statement.

For more information on this statement and the writing of CLISTs, see *CLISTs: Implementation and Reference.*

File variables are only scanned once (no rescans) and only on OPENFILE.

```
[label:]              CLOSFILE   filename
```

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

filename
specifies the DD name by which the file was allocated and opened (by OPENFILE).

CONTROL CLIST Statement

Use the CONTROL statement to define certain processing options to be in effect for the CLIST. The options are in effect from the time CONTROL executes until either the CLIST terminates or another CONTROL is issued.

CLISTs without CONTROL statements execute with options MSG, TRANS, NOLIST, NOPROMPT, NOCONLIST, NOSYMLIST, and FLUSH. You can set PROMPT and LIST by entering them as operands on the EXEC command or subcommand that invokes the CLIST.

CONTROL has no default operands. If you enter CONTROL with no operands, the system uses options already in effect because of system pre-definition, presetting by EXEC, or setting by a previous CONTROL statement. In addition, if there are no operands specified, the system displays those options which are currently in effect.

For more information on this statement and the writing of CLISTs, see *CLISTs: Implementation and Reference.*

Note: Do not enter CONTROL operands as symbolic variables.

```
[label:]  CONTROL  [CONLIST  ]
                   [NOCONLIST]

                   [FLUSH   ]
                   [NOFLUSH ]

                   [LIST    ]
                   [NOLIST  ]

                   [MSG     ]
                   [NOMSG   ]

                   [PROMPT  ]
                   [NOPROMPT]

                   [SYMLIST ]
                   [NOSYMLIST]

                   [CAPS      ]
                   [NOCAPS/ASIS]

                   [MAIN]
                   [END(string)]
```

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

CONLIST
specifies CLIST statements are displayed at the terminal after symbolic substitution, but before execution.

NOCONLIST
specifies CLIST statements are not displayed at the terminal.

FLUSH
specifies the system can purge (flush) the queue called the input stack. The system normally flushes the stack on an execution error.

NOFLUSH
specifies the system cannot flush the stack.

LIST
specifies commands and subcommands are displayed at the terminal after symbolic substitution, but before execution.

NOLIST
specifies commands and subcommands are not displayed at the terminal.

MSG
specifies PUTLINE informational messages from commands and statements in the procedure are displayed at the terminal.

NOMSG
specifies PUTLINE informational messages NOMSG from commands and statements in the procedure are not displayed at the terminal.

PROMPT
specifies the CLIST can prompt the terminal for input.

NOPROMPT
specifies the CLIST cannot prompt the terminal for input, even if the procedure has prompting capabilities.

SYMLIST
specifies executable statements are displayed at the terminal once before the scan for symbolic substitution. Executable statements include commands, subcommands, and CLIST statements.

NOSYMLIST
specifies executable statements are not displayed at the terminal before symbolic substitution.

CAPS
indicates input strings are translated to uppercase letters before being processed.

NOCAPS/ASIS
indicates input strings are not translated to uppercase letters before being processed.

MAIN
specifies this is the main CLIST in your TSO environment. The system cannot delete MAIN by a stack flush. If you specify MAIN, the system ignores FLUSH and NOFLUSH. The attention exit in the TMP cannot delete the CLIST and any error exit used by this CLIST is protected.

END(string)

specifies a character string be recognized by the system as an END statement that concludes a DO-group. Enter the string as 1-4 characters, the first alphabetic and the rest alphameric. Because END no longer signifies the end of a DO-group, the writer of the CLIST can include END commands and subcommands without prematurely ending the DO-group.

DATA-ENDDATA CLIST Sequence

Use the DATA and ENDDATA statements to designate a group of commands and subcommands that are looked at as data by the CLIST, but as commands and subcommands by the system. Symbolic substitution is performed before execution of the group. CLIST statements included in the DATA-ENDDATA group might cause failures because TSO attempts to execute them as commands or subcommands. A DO-group ignores an END in an included DATA-ENDDATA group, instead of terminating the DO-group.

For more information on this statement and the writing of CLISTs, see *CLISTs: Implementation and Reference*.

```
[label:]              DATA
                        .
                        .
                        .
                        .
                      ENDDATA
```

label:
specifies a name to which the CLIST can branch. Enter one-to-eight characters, the first alphabetic and the rest alphameric, followed by a colon. You cannot specify a label for ENDDATA.

Example

Operation: Perform an EDIT operation without ending a DO-group.

```
.
.
IF &ADDIT=YES THEN -
   DO
        DATA
            EDIT OLD.DATA
            BOTTOM
            INSERT * &NEW ENTRY
            END SAVE
        ENDDATA
        .
        .
   END
ELSE
.
.
```

DATA PROMPT-ENDDATA Sequence

Use the DATA PROMPT-ENDDATA sequence to designate a group of lines within a CLIST as replies when the system prompts for data. An error condition (error code 968) occurs unless each DATA PROMPT statement is immediately preceded by a command or subcommand that issues a prompt. If a prompt occurs, you receive a reply.

For more information on this sequence and the writing of CLISTs, see *CLISTs: Implementation and Reference.*

```
[label:]              DATA PROMPT
                        .
                        .
                        .
                        .
                      ENDDATA
```

label:
specifies a name to which the CLIST can branch.

There are certain rules to remember when using the DATA PROMPT-ENDDATA sequence. They are:

- The CLIST must allow prompting with the CONTROL PROMPT or PROMPT operand on the EXEC command.
- Symbolic substitution is performed before a reply is sent.
- Do not enter the PROMPT operand as a symbolic variable.
- The DATA PROMPT sequence is sensitive to the following types of prompts:

```
PUTGET PROMPT
PTBYPASS
TERM
ATTN
GETLINE TERM
```

Example

Operation: Use the DATA PROMPT feature to supply EDIT with input in either foreground or background.

Procedure:

```
CONTROL PROMPT
.
.          DEFINES A NULL VARIABLE
SET & NULL = R
EDIT EXAMPLE.DATA NEW
DATA PROMPT/*SUPPLY REPLIES TO INPUT PROMPT*/
THIS WILL BE LINE 1
THIS WILL BE LINE 2
&NULL
ENDDATA
END SAVE
```

Result: The contents of the created data set will be:

```
00010     THIS WILL BE LINE 1
00020     THIS WILL BE LINE 2
```

DO-WHILE-END CLIST Sequence

Use the DO, WHILE, and END statements to form commands, subcommands, and statements into DO-groups of related instructions. DO and END denote the start and end, respectively, of the DO-group. WHILE specifies a condition and causes the DO-group to re-execute as long as the condition is true.

You can use the string specified on the END operand of the CONTROL statement instead of the END statement.

For more information on these statements and the writing of CLISTs, see *CLISTs: Implementation and Reference.*

```
[label:]        DO  [WHILE logical-expression]
                  .
                  .
                  .
                  .
[label:]        END
```

label:
specifies a name to which the CLIST can branch. Enter one-to-eight characters, the first alphabetic and the rest alphameric, followed by a colon.

logical-expression
is a set group of comparative expressions grouped by logical operators. The minimal entry for logical-expression is a comparative expression.

ERROR CLIST Statement

Use the ERROR statement to check for nonzero (error-condition) return codes from commands, subcommands, and CLIST statements in the currently executing CLIST. When an error code is detected, the system effectively performs an error exit.

The error exit must be protected from being flushed from the input stack by the system. Stack flushing makes the error return codes unavailable. Use the MAIN or NOFLUSH operands of the CONTROL statement to prevent stack flushing.

If you specify ERROR with no operands, the system displays any command, subcommand, or statement in the CLIST that ends in error. The system then attempts to continue with the next sequential statement, if possible.

Note the following:

- If the LIST option was requested for the CLIST being executed, the NULL error statement is ignored.
- The ERROR statement must precede any statements that might cause a branch to it.

For more information on this statement and the writing of CLISTs, see *CLISTs: Implementation and Reference*.

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

---

label:
specifies a name to which the CLIST can branch. Enter one-to-eight characters, the first alphabetic and the rest alphameric, followed by a colon.

OFF
specifies any action previously set up by an ERROR statement is nullified. Note that OFF is not a default.

action
specifies any executable statement, commonly a DO-group constituting a routine.

Note: If only the TIME command or the NULL error statement executes in a DO-group, a recursive CLIST error occurs and an error message is issued. The error occurs because neither the TIME command nor the NULL error statement resets LASTCC.

Example

Operation: Perform an error analysis routine whenever an error occurs in the CLIST.

```
.
.
ERROR   DO
        .
        . /* Error analysis routine */
        .
        END
.
.
```

EXIT CLIST Statement

Use the EXIT statement to return control to the routine that called the currently executing CLIST. You can specify the return code associated with this exit or allow it to default to the value in control variable &LASTCC.

A procedure that is called by another procedure is said to be nested. A called procedure can also call a procedure, which would be considered to be nested two levels. Levels of nesting are limited only by the extent of storage and the skill of the programmer. The structure of the nesting is called the hierarchy. You go up in the hierarchy when control passes from the called to the calling procedure. TSO itself is at the top.

Entering EXIT causes control to go up one level. When EXIT is entered with the QUIT operand, the system attempts to pass control upward to the first procedure encountered that has MAIN or NOFLUSH in effect (see CONTROL statement). If no such procedure is found, control passes up to TSO, the input stack is flushed of all CLISTs, and control passes to the terminal.

For more information on this statement and the writing of CLISTs, see *CLISTs: Implementation and Reference*.

```
[label:]          EXIT    [CODE(expression)]
                          [QUIT]
```

---

label:
specifies a name to which the CLIST can branch. Enter one-to-eight characters, the first alphabetic and the rest alphameric, followed by a colon.

CODE(expression)
specifies a user-defined return code for this exit, with the code specifiable in most simple form as a number or in most complex form as a simple expression. If you do not specify CODE, the system uses the contents of &LASTCC.

QUIT
specifies control is passed up the nested hierarchy until a procedure is found with the MAIN or NOFLUSH option active or until TSO receives control.

GETFILE CLIST Statement

Use the GETFILE statement to get a record from an open QSAM file. One record is obtained for one execution of GETFILE. You must know the filename(ddname) by which you allocated and opened (by OPENFILE) the file for this terminal session.

After GETFILE executes, the file variable, *filename*, contains the record obtained.

File variables are scanned only once (no rescans) and only on OPENFILE.

For more information on this statement and the writing of CLISTs, see *CLISTs: Implementation and Reference*.

```
[label:]          GETFILE     filename
```

label:
specifies a name to which the CLIST can branch. Enter one-to-eight characters, the first alphabetic and the rest alphameric, followed by a colon.

filename
specifies the DD name by which the file was allocated and opened (by OPENFILE).

GLOBAL CLIST Statement

Use the GLOBAL statement to define unique symbolic variables that are to be used globally, which in the application means in all lower nested levels of the hierarchy. The GLOBAL statement must precede any statement that uses its variables. The first level CLIST defines global variables. Lower level procedures must include a GLOBAL statement, if they intend to refer to the global variables specified in the first level. The number of global variables defined in the first level procedure is the maximum number that can be referenced by any lower level procedure.

The global variables are positional, both in the first level procedure and in all lower level procedures that reference this same set of variables. This means that the nth name on any level GLOBAL statement refers to the same variable, even though the symbolic name at each level may be different. Note, however, that the names must still be unique among those at that level.

For more information on this statement and the writing of CLISTs, see *CLISTs: Implementation and Reference.*

```
[label:]        GLOBAL name1 [name2....nameN]
```

---

label:
specifies a name to which the CLIST can branch. Enter one-to-eight characters, the first alphabetic and the rest alphameric, followed by a colon.

name1-nameN
specify valid symbolic variable names for this procedure.

Example

Operation: Specify a set of global variables for three levels of procedures, where some names are unique to their level.

First level procedure:	GLOBAL	NAME1	NAME2	NAME3	NAME4
Second level procedure:	GLOBAL	FIRST	SECOND	THIRD	
Third level procedure:	GLOBAL	PARM1	PARM2	PARM3	PARM4

Note that &NAME3, &THIRD, and &PARM3 would access the same variable.

GOTO CLIST Statement

Use the GOTO statement to cause an unconditional branch within a CLIST. You cannot branch to another CLIST. If you specify GOTO, control passes to the statement or command that has the label called out as the target.

For more information on this statement and the writing of CLISTs, see *CLISTs: Implementation and Reference.*

```
[label:]          GOTO   target
```

---

label:
specifies a name to which the CLIST can branch. Enter one-to-eight characters, the first alphabetic and the rest alphameric, followed by a colon.

target
specifies either a label or an expression that reduces to a valid label value after symbolic substitution.

Example

Operation: Illustrate branching within a CLIST.

```
BEGIN: SET &RET=NEXT
       GOTO LAB1
NEXT:  WRITENR TWO,
       SET &N=2
       GOTO LAB&N
       .
       .
       .
LAB1:  WRITENR ONE,
       GOTO &RET
LAB2:  WRITE THREE
       EXIT /* ONE,TWO,THREE HAS BEEN WRITTEN +
            TO THE TERMINAL*/
```

IF-THEN-ELSE CLIST Statement

Use the IF-THEN-ELSE sequence to define a condition, to test the truth of that condition, and to initiate an action based on the test results.

Caution: A continuation character is required if the THEN or ELSE statement extends to the next line. If no continuation character is present and no other text is on the same line, the THEN and ELSE are treated like null statements. Use a plus (+) or (-) sign to continue to the next line when writing CLISTs.

For more information on this statement and the writing of CLISTs, see *CLISTs: Implementation and Reference.*

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

---

label:
specifies a name to which the CLIST can branch. Enter one-to-eight characters, the first alphabetic and the rest alphameric, followed by a colon.

logical-expression
is a group of comparative expressions grouped by logical operators. The minimal entry for logical-expression is a comparative expression.

action
specifies an executable statement, which includes commands, subcommands, and CLIST statements. The THEN action is invoked if the IF condition is satisfied. The ELSE action is invoked if the IF condition is not satisfied and ELSE is specified. If the IF condition is not satisfied and ELSE is not specified, control passes to the next sequential statement.

OPENFILE CLIST Statement

Use the OPENFILE statement to open a specific file for QSAM I/O. One execution of OPENFILE opens one file. File variables are scanned only once (no rescans) and only on OPENFILE.

Complete your file I/O on a specific file before you change modes from command to subcommand or vice versa. The system does not support crossmode file I/O, which causes miscellaneous abnormal terminations.

Specify NOFLUSH (see the CONTROL statement) for a CLIST that uses file I/O.

If a system action causes the file to be flushed because you did not specify NOFLUSH, you have to log off the system to recover. You recognize the condition by getting a message similar to "FILE NOT FREED, DATA SET IS OPEN."

For reference information on QSAM I/O, see *Data Management Services.*

For more information on this statement and the writing of CLISTs, see *CLISTs: Implementation and Reference* .

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

label:
specifies a name to which the CLIST can branch. Enter one-to-eight characters, the first alphabetic and the rest alphameric, followed by a colon.

filename
specifies the DD name of a file that has been previously allocated by the ALLOCATE command or by step allocation. The file name becomes a symbolic variable that contains either:

- The results of a GETFILE
- A record that was set by the user for a PUTFILE.

The file name name does not have to be previously defined.

INPUT
specifies that the file name will open for input. The default is INPUT when neither INPUT, OUTPUT, nor UPDATE is entered.

OUTPUT
specifies the file name is to open for output.

UPDATE
specifies the file name is to open for updating in place; that is, you can replace a previously read record by issuing a PUTFILE statement.

PROC CLIST Statement

Use the PROC statement to define the parameters that can be passed to the CLIST by the value list parameter of the EXEC command. PROC is optional for a CLIST, but if it is used, it must be the first statement in the CLIST.

Note that you cannot enter a label for a PROC statement.

For more information on this statement and the writing of CLISTs, see *CLISTs: Implementation and Reference*.

```
PROC        positional-specification
            [positional-parameters]
            [keyword-parameters[(values)]]
```

positional-specification
specifies the number of required positional parameters to be passed. Enter 1-5 decimal digits. If you do not have any, enter 0.

positional-parameters
specifies the positional parameters, in sequence, that require initial values in the value list before the CLIST is invoked. Parse prompts for an initial value if one is not there, except when positional-specification = 0 and no prompting is needed because there are no positional parameters.

Positional parameter names are 1-252 characters, the first alphabetic and the rest alphameric. The values must be character strings *without* delimiters.

keyword-parameters(values)
specify the keyword parameters, either with or without values, that do not require initial values in the value list before the CLIST is invoked.

Keyword parameter names are 1-31 characters, the first alphabetic and the rest alphameric. Keywords without values have nothing appended. Keywords with values have the values enclosed in parentheses and appended to their names. Specify the value as a null entry (keep parentheses), a quoted character string, or a non-quoted character string. A quoted character string can include delimiters. These values are defaults.

Note: All symbolic parameters have an initial value at the time the CLIST begins execution. You can dynamically change the symbolic parameter value by specifying the symbolic parameter name on the READ, SET, or READDVAL statements.

PUTFILE CLIST Statement

Use the PUTFILE statement to put a record into an already open QSAM file. One execution of PUTFILE transfers one record. This record must be initialized each time by an assignment statement such as SET unless you want the same record sent more than once. You must know the filename(ddname) by which you allocated and opened (by OPENFILE) the file for this terminal session.

File variables are scanned only once (no rescans) and only on OPENFILE.

For more information on this statement and the writing of CLISTs, see *CLISTs: Implementation and Reference.*

```
[label:]          PUTFILE   filename
```

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

filename
specifies the DD name by which the file was allocated and opened (by OPENFILE). The record that is put in is the value of the file variable &FILENAME.

Example

Operation: Illustrate typical file I/O.

```
.
.
OPENFILE MYOUTPUT OUTPUT
.
.
SET &MYOUTPUT = TEXT STRING
PUTFILE MYOUTPUT /* TEXT STRING is put to the file */
.
.
```

READ CLIST Statement

Use the READ statement to make terminal user input available to the CLIST as values in symbolic variables. You can name these variables in the READ statement or elsewhere in the CLIST. The READ statement is usually preceded by a WRITE to the terminal to identify the expected input.

For more information on this statement and the writing of CLISTs, see *CLISTs: Implementation and Reference*.

```
[label:]        READ [name1  [name2...nameN]]
```

label:
specifies a name to which the CLIST can branch. Enter one-to-eight characters, the first alphabetic and the rest alphameric, followed by a colon.

Note: If you specify READ without parameter names, TSO reads the value of the terminal input line into &SYSDVAL.

name1-nameN
specify any syntactically valid parameter names. The & prefix is optional. These symbolic parameters need not be previously defined. The parameters are positional in the sense that recognizable values entered by the CLIST user are set sequentially into the names specified here. Recognizable values are:

- A character string
- A quoted string
- A parenthesized string
- A null value, specified by entering two adjacent commas (,,) or two adjacent quotes (' '). Double quotes (") are not acceptable and cause an error message.

You can specify any or all of the types on one READ statement.

READDVAL CLIST Statement

Use the READDVAL statement to cause the current value of &SYSDVAL to be parsed into syntactical words and to assign these words to the symbolic parameters specified on the READDVAL statement. Use &SYSDVAL to obtain the terminal user's response line when a READ statement requests terminal input.

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

The assignment is done sequentially on the parameters in the order they are specified. Parameters not assigned a value default to null values. If there are more words than parameters, the leftover words are not assigned.

For more information on this statement and the writing of CLISTs, see *CLISTs: Implementation and Reference.*

```
[label:]     READDVAL  variable1[variablen]...
```

label:
specifies a name to which the CLIST can branch. Enter one-to-eight characters, the first alphabetic and the rest alphameric, followed by a colon.

variable1[variablen]
specifies any valid variable name. A variable need not have been previously defined.

RETURN CLIST Statement

Use the RETURN statement to return control from an error or attention exit to the statement following the one that ended in error or the one that was interrupted by an attention.

RETURN is valid only when issued from an activated error action range or an activated attention action range from this CLIST. If neither of these conditions exists, RETURN is treated as a no-operation.

For more information on this statement and the writing of CLISTs, see *CLISTs: Implementation and Reference.*

```
[label:]        RETURN
```

label:
specifies a name to which the CLIST can branch. Enter one-to-eight characters, the first alphabetic and the rest alphameric, followed by a colon.

SET CLIST Statement

Use the SET statement to assign a specified value to a specified symbolic variable name. One value is assigned to one variable for one execution of SET. You do not need to predefine the variable elsewhere.

You cannot set a variable that is a built-in function.

For more information on this statement and the writing of CLISTs, see *CLISTs: Implementation and Reference.*

```
[label:]     SET symbolic-variable-name  {=}   expression
                                         {EQ}
```

label:
specifies a name to which the CLIST can branch. Enter one-to-eight characters, the first alphabetic and the rest alphameric, followed by a colon.

symbolic-variable-name
specifies the syntactically valid symbolic variable or allowable control variable that is to be set.

EQ or =
specifies the comparison operator EQUAL.

expression
specifies a simple expression as explained in *CLISTs: Implementation and Reference.*

TERMIN CLIST Statement

Use the TERMIN statement to pass control from the CLIST currently executing to the terminal user. TSO prompts the terminal user with a READY message. TERMIN also defines the character strings that a user can enter to return control to the CLIST. A null value can be specified as a character string that the user can enter. TERMIN is usually preceded by a WRITE statement that identifies the expected response to the terminal user.

Control returns to the CLIST at the statement after TERMIN. When control returns, TSO sets &SYSDLM and &SYSDVAL.

For more information on this statement and the writing of CLISTs, see *CLISTs: Implementation and Reference.*

```
[label:]    TERMIN   [string1]   [string2....stringN]
                     [   ,   ]
```

label:
specifies a name to which the CLIST can branch. Enter one-to-eight characters, the first alphabetic and the rest alphameric, followed by a colon.

string1-stringN
specify character strings that the terminal user can enter to return control to the command processor. The &SYSDLM control variable contains the number of the string entered (1 for string1, 2 for string2, etc.) and &SYSDVAL contains the balance of the entered line.

,
specifies the terminal user can enter a null line to return control to the CLIST. Use it only in the first string position.

WRITE and WRITENR Statements

Use the WRITE and WRITENR statements to send text to the terminal user from the CLIST. You can use the text for messages, information, or prompting.

For more information on this statement and the writing of CLISTs, see *CLISTs: Implementation and Reference*.

```
[label:]        WRITE[NR]   text
```

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

WRITE
specifies the cursor move to a new line after the text is displayed.

WRITENR
specifies the cursor does not move to a new line after the text is displayed.

text
specifies what is to be sent to the terminal. You can enter any character string, including symbolic variables. Data enclosed within /* and */ delimiters is also sent to the terminal, even though it might appear as a comment.

Example

Operation: Illustrate WRITE and WRITENR usage.

```
.
.
WRITENR ONE
WRITENR TWO/
WRITENR THREE
WRITE FOUR
WRITE FIVE
.
.
```

The display at the terminal shows:

```
ONETWO/THREEFOUR
FIVE
```

Quick Reference Guide to Commands and Statements

Figure 13 lists TSO commands and CLIST statements in alphabetical order.

COMMAND or STATEMENT	USAGE
ALLOCATE command	To dynamically allocate data sets.
ATTN statement	To set attention routines in a CLIST.
ATTRIB command	To build a list of attributes for non-VSAM data sets.
CALL command	To load and execute a load module.
CANCEL command	To halt the processing of batch jobs submitted at your terminal.
CLOSFILE statement	To close a file in a CLIST.
CONTROL statement	To define processing options for a CLIST.
DATA-ENDDATA statement	To designate a group of commands and subcommands as data for a CLIST.
DATA PROMPT-ENDDATA statement	To designate a group of lines within a CLIST as the prompt replies for commands and subcommands.
DELETE command	To delete data set entries or members of a partitioned data set.
DO-WHILE-END statement	To form commands, subcommands, and statements into DO-groups in a CLIST.
EDIT command	To create, modify, store, submit, retrieve, and delete data sets. See command definitions for definitions of EDIT subcommands.
END command	To end a CLIST.
ERROR statement	To set up error routines in a CLIST.
EXEC command	To execute a CLIST.
EXIT statement	To return control to the calling routine in a CLIST.
FREE command	To release previously allocated data sets, change the output of a SYSOUT data set, delete attribute lists, or change data set disposition.
GETFILE statement	To get a record from an open QSAM file in a CLIST.
GLOBAL statement	To define unique symbolic variables for global use in a CLIST.
GOTO statement	To cause an unconditional branch within a CLIST.
HELP command	To obtain information about the function, syntax, and operands of commands and subcommands and information about certain messages.
IF-THEN-ELSE statement	To define a condition, test it, and initiate action based on it within a CLIST.
LINK command	To invoke the linkage editor service program.
LISTALC command	To obtain a list of data sets allocated during the current TSO session.

Figure 13 (Part 1 of 2). TSO Commands and Statements and Their Uses

COMMAND or STATEMENT	USAGE
LISTBC command	To obtain a listing of the contents of the broadcast data set (SYS1.BRODCAST), which contains messages of general interest.
LISTCAT command	To list entries from a catalog by name or entry type.
LISTDS command	To display the attributes of specific data sets at your terminal.
LOADGO command	To load a compiled or assembled program into real storage and begin execution.
LOGOFF command	To end your terminal session.
LOGON command	To start your terminal session.
OPENFILE statement	To open a specific file for QSAM I/O in a CLIST.
OUTPUT command	To direct output from a job to your terminal or to a specific data set; to delete the output, change output class, route output to a remote work station, or release the output for a job for printing by the subsystem.
PROC statement	To define parameters to be passed to a CLIST using the value list parameter of the EXEC command.
PROFILE command	To establish, change, or list your user profile.
PROTECT command	To prevent unauthorized access to your non-VSAM data sets.
PUTFILE statement	To put a record into an already open QSAM file.
READ statement	To make terminal user input available to the CLIST as values in symbolic variables.
READDVAL statement	To cause the current value of &SYSDVAL to be parsed into syntactical words and assign those words to the symbolic parameters specified on the READDVAL statement.
RECEIVE command	To retrieve transmitted files and restore them to their original format.
RENAME command	To change the name of a non-VSAM cataloged data set, to change the member name of a partitioned data set, or to create an alias for a partitioned data set member.
RETURN statement	To return control from an error range or attention range to the statement following the one that ended in error or that was interrupted by an attention.
RUN command	To compile, load, and execute the source statements in a data set.
SEND command	To send a message to another terminal user or to the system operator.
SET statement	To assign a specified value to a specified symbolic variable in a CLIST.
STATUS command	To display the status of batch jobs at your terminal.
SUBMIT command	To submit one or more batch jobs for processing.
TERMIN statement	To pass control from the CLIST currently executing to the terminal user.
TERMINAL command	To define the operating characteristics of your terminal.
TEST command	To test a program or command processor written in Assembler language.
TRANSMIT command	To send information, such as a message or a copy of information in a data set, to another user in the network.
TSOEXEC command	To invoke an authorized command from an unauthorized environment.
WHEN command	To test 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.
WRITE statement	To send text to a terminal user from a CLIST.

Figure 13 (Part 2 of 2). TSO Commands and Statements and Their Uses

Index

*

* operand
  CHANGE subcommand of EDIT 71
  COPY subcommand of EDIT 79
  DELETE subcommand of EDIT 87

%

% indirection symbol 233
% operand of EXEC command 135

?

? (question mark) 233
? indirection symbol 233

A

abbreviating keyword operands 3
abend occurrences outside home address space 238
abend, recovering 66
abnormal termination
  recovering an EDIT work file 64
absolute address 231
AC operand of LINK command 154
access to storage by TEST 238
ACCODE operand
  ALLOCATE command 17
address operand
  AT subcommand of TEST 251
  CALL subcommand of TEST 256
address operand, RUN subcommand of TEST 312
addressing considerations 236
address1 operand
  AND subcommand of TEST 244
address2 operand
  AND subcommand of TEST 244
ALIAS operand
  DELETE command 47
  LISTCAT command 163
  RENAME command 208
ALIGN operand
  ALLOCATE command 26
ALL operand
  CHANGE subcommand of EDIT 72
  HELP command 144
  LISTCAT command 163
ALLOCATE command 8
  operands
    ACCODE 17
    ALIGN 26
    ALTFILE 16
    AVBLOCK 15
    BFALN 21
    BFTEK 22
    BLKSIZE 15
    BLOCK 14
    BUFL 20
    BUFNO 20
    BUFOFF 23
    BURST 25
    CATALOG 19
    CHARS 25
    COPIES 25
    CYLINDERS 15
    DELETE 19
    DEN 24
    DEST 16
    DIAGNS 23
    DIR 16
    DSORG 23
    DUMMY 11
    EROPT 22
    EXPDT 21
    FCB 26
    FILE or DDNAME 12
    FLASH 25
    HOLD 17
    image-id 26
    INPUT 21
    KEEP 19
    KEYLEN 24
    LABEL 17
    LIKE 18
    LIMCT 23
    LRECL 20
    MAXVOL 17
    MOD 13
    MODIFY 25
    MSVGP 14
    NCP 21
    NEW 13
    NOHOLD 17
    OLD 12
    OPTCD 21
    OUTPUT 21
    PARALLEL 17
    POSITION 17
    PRIVATE 18
    PROTECT 24
    RECFM 22
    RELEASE 19
    RETPD 21
    REUSE 16
    ROUND 19
    SHR 12
    SPACE 14
    SYSOUT 13

TRACKS 15
UCOUNT 17
UNCATALOG 20
UNIT 17
USING 19
VERIFY 26
VOLUME 13
VSEQ 18
ALLOCATE subcommand of EDIT 68
ALLOCATE subcommand of TEST 243
ALLOCATION operand of LISTCAT command 163
ALTFILE operand
ALLOCATE command 16
AMODE operand
CALL subcommand of TEST 257
GO subcommand of TEST 274
LINK command 150
LOADGO command 168
RUN subcommand of TEST 312
AND subcommand of TEST 244
ASIS operand
EDIT command 53
ASM command
EDIT command 51
RUN command 210
assignment of values function of TEST 247
assignment statements (see CLIST statements)
AT subcommand of TEST 251
address 251
address-list 251
COUNT 252
DEFER 252
NOTIFY 252
subcommands-list 252
TITLE 253
attention interruptions in cross-memory mode 238
ATTN statement 340
ATTRIB command 32
ATTRIB subcommand of EDIT 69
ATTRIB subcommand of TEST 255
ATTRILIST operand of FREE command 141
authorized command
running in unauthorized environment 336
automatic qualification 232
AVBLOCK operand of ALLOCATE command 15

B

background behavior of command
CALL 40
LOGOFF 172
LOGON 174
SUBMIT command 218
background processing
checkpointing a data set 64
EDIT command 60
recovering after a terminal line disconnect 67
recovering after an abend 66
recovering after system failure 65
recovering an EDIT work file 64
batch processing
cancelling jobs 42
BEGIN operand
CONTINUE subcommand of OUTPUT 184
OUTPUT command 178
BFALN operand
ALLOCATE command 21
ATTRIB command 36
BFTEK operand
ALLOCATE command 22
ATTRIB command 37
BLKSIZE operand
ALLOCATE command 15
ATTRIB command 34
RECEIVE command 204
BLOCK operand
ALLOCATE command 14
EDIT command 53
RECEIVE command 204
BOTTOM subcommand of EDIT 70
BREAK operand of TERMINAL command 223
BUFL (buffer-length) operand
ALLOCATE command 20
ATTRIB command 34
BUFNO (number-of-buffers) operand
ALLOCATE command 20
ATTRIB command 34
BUFOFF (block-prefix-length) operand
ALLOCATE command 23
ATTRIB command 38
built-in functions 339
BURST operand
ALLOCATE command 25

C

CALL command 40
call operand of LOADGO command 169
CALL subcommand of TEST 256
CANCEL command 42
CANCEL subcommand of TEST 259
CAPS operand
EDIT command 53
CATALOG operand
ALLOCATE command 19
DELETE command 46
LISTCAT command 161
LISTDS command 166
CHANGE subcommand of EDIT 71
changing from one mode to another 63
CHAR operand
PROFILE command 190
TERMINAL command 225
CHARS operand
ALLOCATE command 25
CHECK operand
RUN command 211
RUN subcommand of EDIT 115

checkpointing a data set 64
CKPOINT subcommand of EDIT 77
CLASS operand of OUTPUT command 177
CLEAR operand of TERMINAL command 224
CLIST statements 339
  assignment
    GLOBAL statement 354
    READDVAL statement 361
    SET statement 363
  built-in functions 339
  conditional
    DO-WHILE-END sequence 349
    IF-THEN-ELSE statement 356
  control
    ATTN statement 340
    CONTROL statement 343
    DATA-ENDDATA statement 346
    ERROR statement 350
    EXIT statement 352
    GOTO statement 355
    PROC statement 358
    READ statement 360
    RETURN statement 362
    TERMIN statement 364
    WRITE and WRITENR statements 365
  control variables 339
  EDIT command 51
  error codes 339
  executing CLIST, EXEC command 135
  file access
    CLOSFILE statement 342
    GETFILE statement 353
    OPENFILE statement 357
    PUTFILE statement 359
  operators and expressions 339
  symbolic variables 339
CLOSFILE statement 342
CLUSTER operand
  DELETE command 46
  LISTCAT command 162
CN operand of SEND command 215
CNTL operand of EDIT command 51
COBLIB operand
  LINK command 151
  LOADGO command 169
COBOL operand
  EDIT command 51
  RUN command 211
Code and Go FORTRAN
  EDIT command 52
command procedure (see CLIST statements)
comments 4
compiler type, determining 213
considerations
  addressing 236
  cross-memory using TEST 238
  using virtual fetch services 237
  31-bit addressing for TEST 237
context editing 61
CONTINUE subcommand of OUTPUT 184
control password operand of PROTECT command 196
control section tags 326
CONTROL statement 343
control variables 339
COPIES operand
  ALLOCATE command 25
COPY operand of RECEIVE command 205
COPY subcommand of EDIT 79
COPY subcommand of TEST 260
  address1 operand 260
  address2 operand 260
  LENGTH operand 260
  POINTER operand 260
COPYLIST operand of TRANSMIT command 331
count operand
  CHANGE subcommand of EDIT 71
  COPY subcommand of EDIT 79
  DELETE subcommand of EDIT 87
  DOWN subcommand of EDIT 89
  LIST subcommand of EDIT 102
  MOVE subcommand of EDIT 104
  SCAN subcommand of EDIT 121
  UP subcommand of EDIT 132
CP operand of TEST command 228
CREATION operand of LISTCAT command 163
cross-memory considerations using TEST 238
CYLINDER operand
  ALLOCATE command 15
  RECEIVE command 204

D

data encryption (TRANSMIT and RECEIVE) 324
DATA operand
  LISTCAT command 162
  PROTECT command 197
DATA PROMPT-ENDDATA sequence 347
data set disposition 63
data set, checkpointing 64
DATA-ENDDATA statement 346
data-set-name operand
  EXEC command 135
  LINK command 149
  LISTDS command 165
  LOADGO command 168
  PROTECT command 196
  SAVE subcommand of EDIT 118
  TEST command 227
DATASET or DSNAME operand 11
DC operand of LINK command 153
DCBS operand of LINK command 154
DDNAME operand
  FREE command 141
  TRANSMIT command 331
DELETE command 44
DELETE operand
  ALLOCATE command 19
  FREE command 141

OUTPUT command 179
PROTECT command 197
RECEIVE command 205
DELETE subcommand of EDIT 87
DELETE subcommand of TEST 263
delimiters 4
DEN operand
ALLOCATE command 24
ATTRIB command 38
DEST operand
ALLOCATE command 16
FREE command 141
OUTPUT command 179
determining compiler type 213
DIAGNS (TRACE) operand
ALLOCATE command 23
ATTRIB command 38
DIR operand of ALLOCATE command 16
DIRECTORY operand of RECEIVE command 204
DISCONNECT operand of LOGOFF command 172
disconnection, recovering data 67
DO-WHILE-END sequence 349
DOWN subcommand of EDIT 89
DROP subcommand of TEST 264
DSNAME operand
ALLOCATE command 11, 12
CALL command 41
EDIT command 49
FREE command 141
RECEIVE command 203
TRANSMIT command 330
DSORG operand
ALLOCATE command 23
ATTRIB command 38
DUMMY operand
ALLOCATE command 11

E

EDIT command
subcommands
ALLOCATE 68
ATTRIB 69
BOTTOM 70
CHANGE 71
CKPOINT 77
COPY 79
DELETE 87
DOWN 89
EDIT 95
END 90
FIND 92
FREE 94
INPUT 96
INSERT 98
insert/replace/delete function 100
LIST 102
MOVE 104
PROFILE 112
RENUM 113
RUN 115
SAVE 118
SCAN 121
SEND 123
SUBMIT 124
TABSET 128
TOP 130
UNNUM 131
UP 132
VERIFY 133
edit mode 61
EDIT work file, recovering 64
EMODE operand of EDIT command 49
ENCIPHER operand of TRANSMIT command 331
encryption, data (TRANSMIT and RECEIVE) 324
END command 134
END operand
RECEIVE command 205
WHEN command 337
END subcommand of EDIT 90
END subcommand of OUTPUT 186
END subcommand of TEST 265
end-line-number operand
SAVE subcommand of EDIT 118
ending the EDIT command 64
ENTRIES operand of LISTCAT command 161
EP operand of LOADGO command 170
EPILOG operand of TRANSMIT command 331
EQUATE operand
address 266
data-type 267
LENGTH 267
MULTIPLE 267
symbol 266
EQUATE subcommand of TEST 266
ERASE operand of DELETE command 46
EROPT operand
ALLOCATE command 22
ATTRIB command 37
error codes 339
error messages, COPY subcommand of EDIT 81
ERROR statement 350
EXEC command 135
EXEC subcommand of EDIT 91
EXEC subcommand of TEST 269
executing user-written programs 64
EXIT statement 352
EXPDT (year-day) operand
ALLOCATE command 21
ATTRIB command 36
EXPIRATION operand of LISTCAT command 163
explicit form of EXEC command 135
extended implicit form of EXEC 135

F

failure (system), recovering data 65
FCB operand of ALLOCATE command 26

FIB commands (see foreground-initiated-background commands)
file access statements (see CLIST statements)
FILE operand of FREE command 141
filename operand of CLOSFILE statement 342
FIND subcommand of EDIT 92
FLASH operand of ALLOCATE command 25
floating-point registers 232
foreground-initiated-background (FIB) commands
  CANCEL 42
  OUTPUT 177
  STATUS 217
  SUBMIT 218
forms control buffer (FCB) 26
FORT operand
  RUN command 211
FORTGI operand
  EDIT command 51
  FORTRAN IV (G1) 51
FORTH operand
  EDIT command 51
  FORTRAN IV (H) 51
FORTLIB operand
  LINK command 151
  LOADGO command 169
FORTRAN (H) compiler 121
FORTRAN IV (G1) 51
FORTRAN IV (H) EXTCOMP statements 51
FREE command 140
FREE subcommand of EDIT 94
FREEMAIN subcommand of TEST 270
FULLSCREEN
  logon 173
  operand of TRANSMIT command 331
FUNCTION operand of HELP command 144

G

general registers 232
GENERATIONDATAGROUP operand
  DELETE command 47
  LISTCAT command 163
GETFILE statement 353
GETMAIN
  operands of TEST
    EQUATE 272
    integer 272
    LOC (ANY) 272
    LOC (BELOW) 272
    LOC (RES) 272
    SP 272
GETMAIN subcommand of TEST 272
GLOBAL statement 354
GO operand of RUN command 212
GO subcommand of TEST 274
GOFORT operand
  Code and Go FORTRAN 52
  EDIT command 52
  RUN command 211
GOTO statement 355
GROUP operand of LOGON command 176

H

halt processing of batch jobs 42
HELP command 144
help information 145
HELP subcommand of EDIT 95
HELP subcommand of OUTPUT 187
HELP subcommand of TEST 276
HELP, using 5
HERE operand
  CONTINUE subcommand of OUTPUT 184
  OUTPUT command 178
HISTORY operand
  LISTALC command 157
  LISTCAT command 163
  LISTDS command 165
HOLD operand
  ALLOCATE command 17
  FREE command 141
  LOGOFF command 172
  OUTPUT command 178

I

I operand, INPUT subcommand of EDIT 96
IF-THEN-ELSE statement 356
image-id of ALLOCATE command 26
IMODE operand of EDIT command 50
implicit form of EXEC command 135
INCR operand
  COPY subcommand of EDIT 79
  MOVE subcommand of EDIT 104
increment operand
  INPUT subcommand of EDIT 96
  RENUM subcommand of EDIT 113
INDDNAME or INFILE operand of RECEIVE command 202
INDEX operand of LISTCAT command 162
indirect symbols
  % (percent sign) 233
  definition and use 233
  examples of indirect addressing 233
INDSNAME or INDATASET operand of RECEIVE command 202
input mode
  background behavior 60
INPUT operand
  ALLOCATE command 21
  ATTRIB command 35
INPUT subcommand of EDIT 96
INSERT subcommand of EDIT 98
insert/replace/delete function of EDIT 100
INTERCOM operand of PROFILE command 191

J

JES3 users 179

K

KEEP operand
  ALLOCATE command 19
  FREE command 141
  OUTPUT command 178
KEYLEN operand
  ALLOCATE command 24
  ATTRIB command 39
keyword operands 1, 3

L

LABEL operand
  ALLOCATE command 17
  LISTDS command 165
length operand
  AND subcommand of TEST 244
LET operand of LINK command 152
LEVEL operand
  LISTCAT command 162
  LISTDS command 166
LIB operand
  LINK command 150
  LOADGO command 168
  RUN command 211
  RUN subcommand of EDIT 115
LIKE operand of ALLOCATE command 18
LIMCT (search-number) operand
  ALLOCATE command 23
  ATTRIB command 38
line continuation 4, 59
line mode logon 173
line numbers 58
LINE operand of TRANSMIT command 332
line-number editing 61
line-number operand
  INPUT subcommand of EDIT 96
line-number-1 operand
  CHANGE subcommand of EDIT 71
  DELETE subcommand of EDIT 87
  LIST subcommand of EDIT 102
  SCAN subcommand of EDIT 121
line-number-2 operand
  CHANGE subcommand of EDIT 71
  DELETE subcommand of EDIT 87
  LIST subcommand of EDIT 102
  SCAN subcommand of EDIT 121
line1 operand
  COPY subcommand of EDIT 79
  MOVE subcommand of EDIT 104
line2 operand
  COPY subcommand of EDIT 79
  MOVE subcommand of EDIT 104
line3 operand
  COPY subcommand of EDIT 79
  MOVE subcommand of EDIT 104
line4 operand
  COPY subcommand of EDIT 80
  MOVE subcommand of EDIT 104
LINK command 148
LINK subcommand of TEST 277
LIST command
  operands
    address 278
    address list 278
    data-type 279
    LENGTH 279
    MULTIPLE 280
    PRINT 280
LIST subcommand of EDIT 102
LIST subcommand of TEST 278
LISTALC command 156
LISTALC subcommand of TEST 283
LISTBC command 159
LISTBC subcommand of TEST 284
LISTCAT command 160
LISTCAT subcommand of TEST 285
LISTDCB subcommand of TEST 286
  operands
    address 286
    FIELD 286
    PRINT 287
LISTDEB subcommand of TEST
  operands
    address 288
    FIELD 288
    PRINT 289
LISTDS command 164
LISTDS subcommand of TEST 291
LISTMAP subcommand of TEST 292
LISTPSW operands of TEST
  ADDR 294
  PRINT 294
LISTPSW subcommand of TEST 294
LISTTCB subcommand of TEST
  operands
    ADDR 296
    FIELD names 296
    PRINT 297
LMSG operand
  RUN subcommand of EDIT 115
LOAD subcommand of TEST 299
LOADGO command
  description 167
  operands
    CALL 169
    COBLIB 169
    data-set-list 168
    FORTLIB 169
    LIB 168
    MAP 169
    NAME 170
    NOCALL 170

NOMAP 169
NOPRINT 168
NORES 169
NOTERM 169
PLIBASE 169
PLICMIX 169
PLILIB 169
PRINT 168
RES 169
TERM 169
LOG operand of TRANSMIT command 332
LOG(ALL) operand of TRANSMIT command 332
logging function of TRANSMIT and RECEIVE 324
LOGNAME operand of TRANSMIT command 332
LOGOFF command 172
LOGON command 173
LOGON, fullscreen 173
LPREC operand
RUN command 212
LRECL (logical-record-length) operand
ALLOCATE command 20
ATTRIB command 35
EDIT command 53

M

MAIL operand
LISTBC command 159
LOGON command 175
MAP operand of LINK command 151
MAXVOL operand of ALLOCATE command 17
MEMBERS operand
LISTALC command 157
LISTDS command 165
TRANSMIT command 332
MESSAGE operand of TRANSMIT command 331
messages
COPY subcommand of EDIT 81
MOVE subcommand of EDIT 106
MOD operand of ALLOCATE command 13
MODE operand of PROFILE command 192
modes of operations
changing from one mode to another 63
checkpointing a data set 64
edit mode 61
input mode 58
line continuation 59
line numbers 58
recovering an EDIT work file 64
recovering data 67
syntax checking 59
terminating the EDIT command 64
MODIFY operand of ALLOCATE command 25
module name 231
MOVE subcommand of EDIT 104
MSGID (list) operand
HELP command 145
PROFILE command 192
MSVGP operand of ALLOCATE command 14
MVS/XA
ALLOCATE subcommand of TEST 243
AMODE operand of LINK command 150
AMODE operand of LOADGO command 168
AND subcommand of TEST 244
ATTRIB subcommand of TEST 255
CANCEL subcommand of TEST 259
indirect addressing 233
LINK subcommand of TEST 277
LISTALC subcommand of TEST 283
LISTBC subcommand of TEST 284
LISTCAT subcommand of TEST 285
OR subcommand of TEST 303
PROFILE subcommand of TEST 306
PROTECT subcommand of TEST 307
question mark (?) 233
RENAME subcommand of TEST 311
RMODE operand of LINK command 150
RMODE operand of LOADGO command 168
RUN subcommand of TEST 312
SEND subcommand of TEST 314
STATUS subcommand of TEST 315
SUBMIT subcommand of TEST 316
TERMINAL subcommand of TEST 317
UNALLOC subcommand of TEST 318
31-bit addressing considerations for TEST 237

N

name operand of LISTCAT command 163
NAMES data set
control section tags 326
function 325
nicknames section tags 328
NCAL operand of LINK command 151
NCP operand
ALLOCATE command 21
ATTRIB command 35
NE operand of LINK command 153
NEW operand
ALLOCATE command 13
EDIT command 50
new-line-number operand
RENUM subcommand of EDIT 113
SAVE subcommand of EDIT 118
new-name operand of RENAME command 208
NEW/OLD/MOD/SHR operand of RECEIVE command 204
NEWCLASS operand of OUTPUT command 179
NEXT operand
CONTINUE subcommand of OUTPUT 184
nicknames section tags 328
NODC operand of LINK command 154
NOEPILOG operand of TRANSMIT command 331
NOERASE operand of DELETE command 46
NOGO operand
RUN command 212
RUN subcommand of EDIT 115
NOHOLD operand

ALLOCATE command 17
FREE command 141
OUTPUT command 179
NOINTERCOM operand of PROFILE command 192
NOKEEP operand of OUTPUT command 178
NOLET operand
LINK command 152
LOADGO command 170
NOLINE operand of PROFILE command 191
NOLINES operand of TERMINAL command 223
NOLIST operand
EXEC command 136
LINK command 152
NOLOG operand of TRANSMIT command 332
NOMAIL operand of LISTBC command 159
NOMAP operand
LINK command 151
LOADGO command 169
NOMODE operand of PROFILE command 192
NOMSGID operand of PROFILE command 192
non-VSAM data sets 6
NONCAL operand of LINK command 151
NONE operand of LINK command 153
NONOTICES operand
LISTBC command 159
LOGON command 175
NONOTIFY
SUBMIT command 221
SUBMIT subcommand of EDIT 126
TRANSMIT command 332
NONUM operand
EDIT command 53
NONVSAM or NVSAM operand
DELETE command 47
LISTCAT command 162
NOOL operand of LINK command 153
NOOVLY operand of LINK command 153
NOPAUSE operand
CONTINUE subcommand of OUTPUT 185
OUTPUT command 178
PROFILE command 192
RUN command 212
RUN subcommand of EDIT 115
nopointer operand
AND subcommand of TEST 245
NOPREFIX operand of PROFILE command 193
NOPREVIEW operand of RECEIVE command 204
NOPRINT
LOADGO command 168
NOPROLOG operand of TRANSMIT command 333
NOPROMPT operand
EXEC command 136
INPUT subcommand of EDIT 96
PROFILE command 191
NOPURGE operand
CANCEL command 42
DELETE command 46
NOPWREAD operand of PROTECT command 197
NORECOVER operand of EDIT command 50
EDIT command 50
NOREFR operand of LINK command 152
NORENT operand of LINK command 153
NORES operand of LOADGO command 169
NOREUS operand of LINK command 152
NOSAVE operand, END subcommand of EDIT 90
NOSCAN operand of EDIT command 52
NOSCRATCH operand of DELETE command 46
DELETE command 47
NOSCTR operand of LINK command 153
NOSECONDS operand of TERMINAL command 223
NOSTORE operand of RUN command 212
NOTERM operand
LINK command 154
LOADGO command 169
NOTEST operand
LINK command 154
RUN command 212
NOTICES operand
LISTBC command 159
LOGON command 175
NOTIFY operand
SUBMIT command 220
SUBMIT subcommand of EDIT 126
TRANSMIT command 332
NOTIMEOUT operand of TERMINAL command 224
NOTRAN operand of TERMINAL command 225
NOWAIT operand of SEND command 215
NOWRITE operand of PROTECT command 197
NOWTPMSG operand of PROFILE command 193
NOXCAL operand of LINK command 152
NOXREF operand of LINK command 152
NUM operand of EDIT command 53

O

OBJECT operand
RUN command 212
TEST command 228
OFF operand
address 301
address list 301
ATTN statement 340
SCAN subcommand of EDIT 121
TABSET subcommand of EDIT 128
VERIFY subcommand of EDIT 133
OFF subcommand of TEST 301
OIDCARD operand of LOGON command 176
OL operand of LINK command 153
OLD operand
ALLOCATE command 12
EDIT command 50
old-line-number operand
SAVE subcommand of EDIT 118
old-name operand of RENAME command 208
ON operand
SCAN subcommand of EDIT 121
TABSET subcommand of EDIT 128
VERIFY subcommand of EDIT 133

OPENFILE statement 357
OPERANDS operand of HELP command 144
operator operand of WHEN command 337
operators and expressions 339
OPT operand of RUN command 211
OPTCD operand
  ALLOCATE command 21
  ATTRIB command 36
OR
  address1 303
  address2 303
  LENGTH 303
  POINTER 304
OR subcommand of TEST 303
OUTDATASET operand of TRANSMIT command 333
OUTDDNAME operand of TRANSMIT command 333
OUTDSNAME operand of TRANSMIT command 333
OUTFILE operand of TRANSMIT command 333
OUTPUT command 177
OUTPUT operand
  ALLOCATE command 21
  ATTRIB command 36
output sequence 180
OUTPUT subcommands 183
OVLY operand of LINK command 153

P

PAGESPACE operand
  DELETE command 47
  LISTCAT command 163
PARALLEL operand of ALLOCATE command 17
parameters operand of TEST command 228
PARM operand
  RECEIVE command 202
  TRANSMIT command 333
password
  DELETE command 46
  EDIT command 49
password data set 200
password operand of PROTECT command 197
PAUSE operand
  CONTINUE subcommand of OUTPUT 184
  OUTPUT command 178
  PROFILE command 189, 192
  RUN command 212
  RUN subcommand of EDIT 115
PDS operand of TRANSMIT command 333
PLI operand
  EDIT command 50
  RUN command 211
PLIBASE operand
  LINK command 151
  LOADGO command 169
PLICMIX operand of LINK command 151
PLILIB operand of LOADGO command 169
pointer operand
  AND subcommand of TEST 244
POSITION operand of ALLOCATE command 17
position operand, FIND subcommand of EDIT 92
POSITIONAL operand of HELP command 145
positional operands 1
PREFIX operand of PROFILE command 192
PREVIEW operand of RECEIVE command 204
PRINT operand
  LINK command 150
  OUTPUT command 178
PRIVATE operand of ALLOCATE command 18
PROC statement 358
procedure name operand of EXEC command 135
PROFILE command 189
PROFILE subcommand of EDIT 112
PROFILE subcommand of TEST 306
programming with TEST using virtual fetch services 237
PROLOG operand of TRANSMIT command 333
PROMPT operand
  EXEC command 136
  INPUT subcommand of EDIT 96
  PROFILE command 191
PROTECT command 196
PROTECT operand of ALLOCATE command 24
PURGE operand
  CANCEL command 42
  DELETE command 46
purging jobs 42
PUTFILE statement 359
PWREAD operand of PROTECT command 197
PWWRITE operand of PROTECT command 197

Q

qualified addresses 231
QUALIFY subcommand of TEST
  operands
    address 308
    module-name.entryname 308
    TCB 308
quoted string notation 72, 92

R

R operand, INPUT subcommand of EDIT 96
READ statement 360
READDVAL statement 361
RECEIVE command
  data encryption function 324
  description 201
  logging function 324
RECFM operand
  ALLOCATE command 22
  ATTRIB command 37
RECONNECT operand of LOGON command 175
record format 22

RECOVER facility operand of PROFILE 190
recovering
  after abend 66
  data after disconnection 67
  data after system failure 65
  EDIT command 50
  EDIT work file 64
REFR operand of LINK command 152
registers
  floating-point 232
  general 232
relative address 231
RELEASE operand
  ALLOCATE command 19
  RECEIVE command 204
RENAME command 208
RENAME subcommand of TEST 311
RENT operand of LINK command 153
RENUM operand, SAVE subcommand of EDIT 118
RENUM subcommand of EDIT 113
REPLACE operand of PROTECT command 197
RES operand of LOADGO command 169
RESTORE operand of RECEIVE command 204
restrictions
  breakpoints for cross-memory applications 238
  internal and external symbols 235
RETPD (number-of-days) operand
  ALLOCATE command 21
  ATTRIB command 36
RETURN statement 362
REUS operand of LINK command 152
REUSE operand of ALLOCATE command 16
RMODE operand
  LINK command 150
  LOADGO command 168
ROUND operand of ALLOCATE command 19
RUN command 210
RUN command, executing user-written programs 64
RUN subcommand of EDIT 115
RUN subcommand of TEST 312

S

SAVE subcommand of EDIT 118
SAVE subcommand of OUTPUT 188
SCAN operand of EDIT command 52
SCAN subcommand of EDIT 121
SCRATCH operand of DELETE command 46
SCRSIZE operand of TERMINAL command 224
SCTR operand of LINK command 153
SECONDS operand of TERMINAL command 223
SEND command 214
SEND subcommand of TEST 314
SEQUENTIAL operand of TRANSMIT
 command 333
service units
  CPU 322
  I/O 322
  storage 322
Session Manager, TSO commands 6
SET statement 363
SHR operand
  ALLOCATE command 12
SIZE operand
  LINK command 153
  RUN command 212
  RUN subcommand of EDIT 115
SMCOPY 6
SMFIND 6
SMPUT 6
SMSG operand
  RUN subcommand of EDIT 115
SNUM operand, LIST subcommand of EDIT 102
SOURCE operand of RUN command 212
source statements, running 210
SPACE operand
  ALLOCATE command 14
  AVBLOCK 14
  BLOCK 14
  CYLINDERS 14
  DELETE command 47
  increment 14
  LISTCAT command 162
  quantity 14
  RECEIVE command 203
  TRACKS 14
SPREC operand
  RUN command 212
statements, CLIST (see CLIST statements)
STATUS command 217
STATUS operand
  LISTALC command 156
  LISTDS command 165
STATUS subcommand of TEST 315
STORE operand
  RUN command 212
  RUN subcommand of EDIT 115
string operand
  CHANGE subcommand of EDIT 71
  COPY subcommand of EDIT 80
  FIND subcommand of EDIT 92
  insert/replace/delete function of EDIT 100
  MOVE subcommand of EDIT 104
subcommands for EDIT 55
SUBMIT
  command 218
  subcommand of EDIT 124
  subcommand of TEST 316
  support in batch 218
symbolic address 231
symbolic variables 339
symbols
  external 235
  internal 236
  restrictions 235
syntax checking 59
syntax notation conventions 2
SYNTAX operand of HELP command 144
SYSNAMES operand of LISTALC command 157

SYSOUT operand
ALLOCATE command 13
RECEIVE command 204
TRANSMIT command 333
SYSRC operand of WHEN command 337
system failure, recovering data 65

T

TABSET subcommand of EDIT 128
tabulation characters 63
tag definitions
control section 326
nicknames section 328
TERM operand
LINK command 154
LOADGO command 169
TERMIN statement 364
TERMINAL command 222
TERMINAL operand of TRANSMIT command 331
TERMINAL subcommand of TEST 317
terminating the EDIT command 64
TEST
addressing considerations 236
addressing conventions 231
command 227
definition of address expression 234
examples of using TEST 228
restrictions on internal and external symbols 235
setting breakpoints for cross-memory applications 238
subcommands
ALLOCATE 243
AND 244
AT 251
ATTRIB 255
CALL 256
CANCEL 259
COPY 260
DELETE 263
DROP 264
END 265
EQUATE 266
EXEC 269
FREEMAIN 270
GETMAIN 272
GO 274
HELP 276
LINK 277
LIST 278
LISTALC 283
LISTBC 284
LISTCAT 285
LISTDCB 286
LISTDEB 288
LISTDS 291
LISTMAP 292
LISTPSW 294
LISTTCB 296
LOAD 299
OFF 301
OR 303
PROFILE 306
PROTECT 307
QUALIFY 308
RENAME 311
RUN 312
SEND 314
STATUS 315
SUBMIT 316
TERMINAL 317
UNALLOC 318
WHERE 319
types of addresses 231
using virtual fetch services 237
valid address examples 236
when to use 230
TEST operand
RUN command 212
TEST subcommands, list of 239
TEXT operand
EDIT command 51
SEND command 214
TIME command 322
TIMEOUT operand of TERMINAL command 224
TOP subcommand of EDIT 130
TRACKS operand
ALLOCATE command 15
RECEIVE command 204
TRAN operand of TERMINAL command 225
TRANSMIT command 323
data encryption function 324
logging function 324
TRTCH operand
ALLOCATE command 24
ATTRIB command 38
TSO command, definition 1
TSO/E Interactive Data Transmission
RECEIVE command 201
TRANSMIT command 323
TSOEXEC command 336

U

UCOUNT operand of ALLOCATE command 17
UNALLOC subcommand of TEST 318
unauthorized environment
running authorized commands 336
UNCATALOG operand
ALLOCATE command 20
FREE command 142
UNIT operand
ALLOCATE command 17
RECEIVE command 203
UNNUM operand, SAVE subcommand of EDIT 118
UNNUM subcommand of EDIT 131
UP subcommand of EDIT 132
USER operand

SEND command 214
USERCATALOG operand of LISTCAT command 162
USERID operand of RECEIVE command 202
using HELP 5
USING operand of ALLOCATE command 19

V

VERIFY operand of ALLOCATE command 26
VERIFY subcommand of EDIT 133
virtual fetch services 237
VOLUME operand
ALLOCATE command 13
LISTCAT command 163
RECEIVE command 203
VSAM data sets 6
VSBASIC
EDIT command 52
RUN command 212
VSEQ operand
ALLOCATE command 18

W

WAIT operand of SEND command 215
WHEN command 337
WHERE subcommand of TEST 319
WRITE and WRITENR statements 365

X

XCAL operand of LINK command 152
XREF operand of LINK command 152

3

31-bit addressing considerations for TEST 237
3211 Printer 26
3800 Printer
BURST/NOBURST operand of ALLOCATE 25
CHARS operand of ALLOCATE 25
COPIES operand of ALLOCATE 25
FCB operand of ALLOCATE 26
FLASH operand of ALLOCATE 25
image-id operand of ALLOCATE 26
MODIFY operand of ALLOCATE 25

TSO Extensions Command Language Reference

SC28-1307-1

S370-39

Printed in U.S.A.

TSO Extensions
Command Language
Reference

SC28-1307-1

READER'S
COMMENT
FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Note: ***Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.***

Possible topics for comment are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If you wish a reply, give your name, company, mailing address, and date:

_____

_____

_____

_____

What is your occupation? _____

How do you use this publication? _____

Number of latest Newsletter associated with this publication: _____

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

Note: Staples can cause problems with automated mail sorting equipment.
Please use pressure sensitive or other gummed tape to seal this form.

Cut or Fold Along Line

TSO Extensions Command Language Reference

SC28-1307-1 S370-39

Reader's Comment Form

Cut or Fold Along Line

Fold and tape Please Do Not Staple Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department D58, Building 921-2
PO Box 390
Poughkeepsie, New York 12602

Fold and tape Please Do Not Staple Fold and tape

Printed in U.S.A.

TSO Extensions
Command Language
Reference

SC28-1307-1

READER'S
COMMENT
FORM

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Note: ***Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.***

Possible topics for comment are:

Clarity Accuracy Completeness Organization Coding Retrieval Legibility

If you wish a reply, give your name, company, mailing address, and date:

_____

_____

_____

_____

Note: Staples can cause problems with automated mail sorting equipment. Please use pressure sensitive or other gummed tape to seal this form.

Cut or Fold Along Line

What is your occupation? _____

How do you use this publication? _____

Number of latest Newsletter associated with this publication: _____

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

TSO Extensions Command Language Reference

SC28-1307-1 S370-39

Reader's Comment Form

Cut or Fold Along Line

Fold and tape Please Do Not Staple Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department D58, Building 921-2
PO Box 390
Poughkeepsie, New York 12602

Fold and tape Please Do Not Staple Fold and tape

IBM®

Printed in U.S.A.

SC28-1307-01