Data General

Addendum

Addendum to Using the CLI (AOS/VS and AOS/VS II)

086–000200–00

This addendum updates your manual 093–000646–01. Please see "Updating Your Manual."

Notice

Addendum to
Using the CLI (AOS/VS and AOS/VS II)
086-000200-00

In the margins of replacement pages, a vertical bar indicates substantive technical change from 093-000646-01.

The addendum number appears on all pages in this addendum.

Updating Your Manual

This addendum (086-000200-00) to *Using the CLI (AOS/VS and AOS/VS II)* introduces new information effective with AOS/VS Release 7.70 and AOS/VS II Release 2.20. It also includes minor corrections. Where new material requires additional pages, the pages have a decimal and number suffix; for example 5-2.1, 5-2.2.

To update your copy of 093-000646-01 please remove manual pages and insert addendum pages as follows:

Remove	**Insert**
Title/Notice for 093-000646-01	Title/Notice for 093-000646-01
iii through vii	iii through vii
old Table of Contents	new Table of Contents
1-5 through 1-14	1-5 through 1-14
1-21 through 1-26	1-21 through 1-26
2-9/2-10	2-9/2-10
3-5 through 3-10	3-5 through 3-10
4-21/4-22	4-21 through 4-22.2
5-3 through 5-20	5-3 through 5-20
5-83 through 5-86	5-83 through 5-86.2
5-93 through 5-102	5-93 through 5-102.2
5-137 through 5-152	5-137 through 5-152.4
5-171/5-172	5-171 through 5-172.4
5-177/5-178	5-177/5-178
5-185 through 5-194	5-185 through 5-194.4
5-203 through 5-206	5-203 through 5-206
5-221/5-222	5-221/5-222
5-229 through 5-232	5-229 through 5-232.2
5-243 through 5-256	5-243 through 5-256.2
5-261 through 5-264	5-261 through 5-264
5-269 through 5-282	5-269 through 5-282.6
5-299 through 5-302	5-299 through 5-302.2
5-315/5-316	5-315/5-316
5-321/5-322	5-321/5-322
5-335 through 5-338	5-335 through 5-338
5-365/5-366	5-365/5-366
5-385 through 5-390	5-385 through 5-390.2
5-401/5-402	5-401 through 5-402.2
5-411/5-412	5-411/5-412
5-445 through 5-456	5-445 through 5-456
5-475/5-476	5-475/5-476
5-487 through 5-490	5-487 through 5-490
5-495 through 5-502	5-495 through 5-502
5-515/5-516	5-515/5-516
old Index	new Index
old Document Set	new Document Set

Insert this sheet immediately behind the new Title/Notice page.

Using the CLI (AOS/VS and AOS/VS II)

093–000646–01

For the latest enhancements, cautions, documentation changes, and other information on this product, please see the Release Notice (085–series) supplied with the software.

Notice

Using the CLI (AOS/VS and AOS/VS II)
093–000646–01

Revision History:	Effective with:
Original Release as 093–000122 – April 1976	
First Revision – April 1977	
Second Revision – June 1978	
Third Revision – June 1979	
Fourth Revision – November 1980	
Fifth Revision – May 1982	
Sixth Revision – February 1984	
Seventh Revision – October 1984	
Original Release as 093–000646 – March 1990	
First Revision – December 1991	
Addendum 086–000200–00 – July 1992	AOS/VS II, Rev. 2.20, AOS/VS, Rev. 7.70

A vertical bar in the margin of a page indicates substantive technical change from the previous revision.

About this Manual

This manual describes the AOS/VS and AOS/VS II Command Line Interpreter (CLI).

This manual is for any user of the CLI. There is no prerequisite in terms of experience, but if you have no experience with the CLI and will use it extensively, you may want to read *Learning to Use Your AOS/VS System* for background. *Learning to Use* leads you through a sample CLI session, briefly explains some common CLI commands, and supplies essential background information about Data General's AOS/VS and AOS/VS II operating systems.

This manual describes some conceptual material, but is designed primarily as reference. After reading about the commands you are interested in, you should be able to use them productively. Errors that return from commands are explained in the manual *AOS/VS and AOS/VS II Error and Status Messages*.

Organization of the Manual

This manual is organized as follows.

Chapter 1	Introduces the CLI, its command syntax, and its on–line help messages.
Chapter 2	Outlines the AOS/VS and AOS/VS II file system, and explains how to navigate through it with the CLI.
Chapter 3	Explains the CLI environment levels and how you can use them.
Chapter 4	Explains CLI macros (files of CLI commands).
Chapter 5	Explains all CLI commands, macros, and pseudomacros alphabetically. This chapter also explains how to use selected utility programs such as BROWSE, DUMP_II and DISPLAY. These utility programs are not documented elsewhere. A later section in this Preface, "Related Manuals," lists the documentation for other utility programs and languages.
Chapter 6	Explains how to use labeled or unlabeled magnetic tape and diskettes.
Appendix A	Contains the ASCII character set.
Appendix B	Contains a description of your terminal keypad.
Appendix C	Explains how to submit a batch job on punched cards.
Appendix D	Describes how to use a Digital Equipment Corporation VT100–class terminal with the CLI.

This manual does not have a separate glossary. Instead, see the manual *AOS/VS and AOS/VS II Glossary*.

We supply tabbed dividers with this manual to help you find important reference material quickly. Instructions for inserting these tabs in your manual appear before this section.

Related Manuals

Chapter 5 of this manual describes those utilities that are not explained in any other manual. Depending on your site's needs and your background, you may find other Data General manuals helpful. This section includes the manuals you may find most useful in conjunction with *Using the CLI*. Refer to the Document Set, located after the Index of this manual, for a list of all AOS/VS and AOS/VS II documentation.

For more information on system management programs like CONTEST, DISCO, EXEC, LDCOPY, LDUINFO, PED, PCOPY, PREDITOR, REPORT, or SPRED, see *Managing AOS/VS and AOS/VS II* and the pertinent Help topic. For information on the Disk Formatter, Disk Jockey, FIXUP, the Installer, POLISHER, PREDITOR, UP.CLI, or VSGEN, see the "Installing" manual for your operating system.

AOS/VS and AOS/VS II Manuals

The following manuals are part of the AOS/VS and AOS/VS II operating system set.

Learning to Use Your AOS/VS System (069–000031)

A primer for all users, this manual introduces AOS/VS (but the material also applies to AOS/VS II) through interactive sessions with the CLI, the SED and SPEED text editors, programming languages, assembler, and the Sort/Merge utility.

AOS/VS and AOS/VS II Glossary (069–000231)

For all users, this manual defines important terms used in AOS/VS and AOS/VS II manuals, for both regular and preinstalled systems.

AOS/VS and AOS/VS II Error and Status Messages (093–000540)

For all users, this manual explains error and status messages that might return from CLI commands, what the messages mean, and how to recover from error conditions.

SED Text Editor User's Manual (AOS and AOS/VS) (093–000249)

For all users, this manual explains how to use SED, an easy–to–use screen–oriented text editor that lets you program function keys to make repetitive tasks easier. The *SED Text Editor* template (093–000361) accompanies this manual.

Managing AOS/VS and AOS/VS II (093–000541)

For system managers and operators, this manual explains managing an AOS/VS or AOS/VS II system. Managing tasks include editing user profiles, managing the multiuser environment with the EXEC program, backing up and restoring files, using runtime tools, and so forth. The manual also includes material of interest to programmers, such as how to write an EXEC cooperative or a custom logon program. This manual complements the "Installing" manuals, for both regular and preinstalled systems.

AOS/VS Debugger and File Editor User's Manual (093–000246)

For assembly language programmers, this manual describes using the AOS/VS and AOS/VS II debugger for examining program files, and the file editor FED for examining and modifying locations in any kind of disk file, including program and text files. The *AOS/VS Debug/FED* template (093–000396) accompanies this manual.

AOS/VS Link and Library File Editor (LFE) User's Manual (093–000245)

For AOS/VS and AOS/VS II programmers, this manual describes the Link utility, which builds executable program files from object modules and library files, and which can also be used to create programs to run under the AOS, MP/AOS, RDOS, RTOS, or DG/UX™ operating systems. This manual also describes the Library File Editor utility, LFE, for creating, editing, and analyzing library files; and the utilities CONVERT and MKABS, for manipulating RDOS and RTOS files.

AOS/VS, AOS/VS II, and AOS/RT32 System Call Dictionary, ?A through ?Q (093–000542)
AOS/VS, AOS/VS II, and AOS/RT32 System Call Dictionary, ?R through ?Z (093–000543)

For system programmers and application programmers who want to use system calls, this two–volume manual set provides detailed information about their use, syntax, accumulator input and output values, parameter packets, and error codes.

Application Development Software Manuals

This manual does not explain commands for using Data General application development products (for example, the SED text editor, FORTRAN 77 compiler, and Link program). The following manuals cover some popular Data General application development products that run under AOS/VS and AOS/VS II.

AOS/VS BASIC Reference Manual (093–000252)
The C Language Reference and Runtime Manual (093–000264)
Addendum to the C Language Reference and Runtime Manual (086–000094)
The C Language Summary (069–000053)
COBOL Reference Manual (AOS/VS) (093–000289)
Data General's FORTRAN (069–000029)
FORTRAN 77 Documentation Summary (069–000080)
FORTRAN 77 Reference Manual (093–000162)
FORTRAN 77 Environment Manual (AOS/VS) (093–000288)
Sort/Merge with Report Writer User's Manual (AOS and AOS/VS) (093–000155)
Pascal (AOS/VS and DG/UX™) Language Summary (069–000037)
Pascal (AOS/VS and DG/UX™) Reference Manual (093–000290)
Plain PL/I (A PL/I Primer) (069–000021)
PL/I Reference Manual (AOS/VS) (093–000270)
Addendum to PL/I Reference Manual (AOS/VS) (086–000067)
SWAT® Summary (AOS/VS) (069–000102)
SWAT® Debugger User's Manual (093–000258)
Addendum to SWAT® Debugger User's Manual (086–000045)
Using the SWAT® Debugger (AOS/VS) (093–000407)

Reader, Please Note:

Within this manual, CLI commands appear in upper– and lowercase; you can type them in lowercase, uppercase, or any combination. All numbers are decimal unless indicated otherwise; for example, 35 (octal).

We use certain symbols in special ways:

Symbol	**Means**
)	The default AOS/VS and AOS/VS II CLI prompt.
Su)	The AOS/VS and AOS/VS II CLI Superuser prompt.
Sp)	The AOS/VS and AOS/VS II CLI Superprocess prompt.
Sm)	The AOS/VS and AOS/VS II CLI System Manager prompt.
□	Be sure to put a space here. (We use this only where we must; normally, you can see where to put spaces.)
↵	Press the NEW LINE, Carriage Return (CR), or Enter key on your terminal keyboard. (If you are using a Digital Equipment Corporation VT100 or VT220 terminal, press the RETURN or Return key.)

When manual text indicates that you should *enter* something, we mean that you should press the NEW LINE, Carriage Return (CR), or Enter key on your terminal's keyboard after typing the appropriate text.

We use these conventions for command formats:

REQUIRED required *[optional]* ...

Where	**Means**
REQUIRED	You must type the uppercase word, such as a command (or its accepted abbreviation), as shown.
required	You must type an argument, filename, or other variable in place of the lowercase word or letter. For example, the x in @MTx0 can be the letter B, C, D, or J, depending on the type of magnetic tape unit.
$\left\{\begin{matrix}\text{required}_1\\ \text{required}_2\end{matrix}\right\}$	You must type *one* of the required arguments. Do not type the braces; they only set off the choice of arguments.
[optional]	You have the option of typing this argument. Do not type the brackets; they only set off what is optional.
$\left[\begin{matrix}\textit{optional}_1\\ \textit{optional}_2\end{matrix}\right]$	You may type *one* of the optional arguments. Do not type the brackets; they only set off the choice of optional arguments.

[...]	You may repeat the preceding entry or entries. The explanation will tell you what you may repeat.

Finally, within examples and descriptive text, we use

THIS TYPEFACE TO SHOW YOUR ENTRY)
This typeface for system queries and responses
`This typeface to show listings`

Contacting Data General

Data General wants to assist you in any way it can to help you use its products. Please feel free to contact the company as outlined below.

Manuals

If you require additional manuals, please use the enclosed TIPS order form (United States only) or contact your local Data General sales representative.

Telephone Assistance

If you are unable to solve a problem using any manual you received with your system, free telephone assistance is available with your hardware warranty and with most Data General software service options. If you are within the United States or Canada, contact the Data General Customer Support Center (CSC) by calling 1–800–DG–HELPS. Lines are open from 8:00 a.m. to 5:00 p.m., your time, Monday through Friday. The center will put you in touch with a member of Data General's telephone assistance staff who can answer your questions.

For telephone assistance outside the United States or Canada, ask your Data General sales representative for the appropriate telephone number.

Joining Our Users Group

Please consider joining the largest independent organization of Data General users, the North American Data General Users Group (NADGUG). In addition to making valuable contacts, members receive *FOCUS* monthly magazine, a conference discount, access to the Software Library and Electronic Bulletin Board, an annual Member Directory, Regional and Special Interest Groups, and much more. For more information about membership in the North American Data General Users Group, call 1–800–253–3902 or 1–508–443–3330.

End of Preface

Contents

Chapter 4 – CLI Macros

Chapter 5 – CLI Commands, Macros, Pseudomacros, and Utility Programs

Tables

Figures

```
) TYPO MINE1    MINE2 <TAB> MINE3)
```

the CLI would return with

Error: Not a command, macro, or program, TYPO
TYPO,MINE1,MINE2,MINE3

A *null* argument has no value, but exists (as opposed to an omitted or missing argument). You can specify the null argument by typing two consecutive commas (,,) or a separator followed by a delimiter (,))

A command line *delimiter* completes a command line and tells the CLI to process it. The CLI recognizes the following characters as command line delimiters: NEW LINE, Form Feed, Carriage Return (CR), and end of file (CTRL–D, CTRL–D). If your terminal does not have a NEW LINE key, use the carriage return key (usually labeled as CR or RETURN) instead.

Correcting Typing Errors

You may make errors as you type commands to the CLI. The last example, which begins with the nonexistent TYPO command, resulted in an error message from the CLI. You can change a command *before* pressing the delimiter that tells the CLI to act on it. For now we give the following two ways:

- Press the CTRL key, hold it down, and press the U key (either uppercase or lowercase). This erases the command and you see the CLI prompt,). You can try again to type a correct command. The letters CTRL–U represent this sequence of pressing keys.
- Press the leftarrow (←) and rightarrow (→) keys to move the cursor left and right. Then strike over incorrect letters with correct ones.

The following examples show correction of errors. The person has not pressed a command line delimiter before correcting the commands.

```
) TYPO  MY_FLE1  HIZFILE22  CTRL–U
```

The trailing CTRL–U sequence erases the entire line (on a CRT terminal). On a hardcopy terminal, you might press NEW LINE after CTRL–U to start again at the left margin. On a CRT, instead of pressing CTRL–U, you could use the arrow keys to correct the existing line: press the ← key until you can overstrike the O in TYPO with E; then press the → key until you can overstrike the FLE1 with FILE.

At any point, you can press CTRL–E to start inserting new characters in a line; pressing CTRL–E again terminates insert mode.

Later in this chapter, under the head "Screenedit Control Characters," you'll see how several other special keystrokes change a CLI command line.

Abbreviating CLI Commands

You can abbreviate CLI commands and command switches. The shortest acceptable abbreviation is the smallest number of characters, beginning with the first character, that uniquely identifies the command or switch.

NOTE: Unlike other command and switch names, the names of global switches /STR= and /ESTR= cannot be abbreviated.

For example, X or XE is a valid abbreviation for the XEQ command because no other command begins with X. DE, however, is not a valid command abbreviation because DEASSIGN, DEBUG, DEFACL, and DELETE all begin with DE. (You could use DEA, DEB, DEF, and DEL, respectively.)

NOTE: When you write a macro (as described in Chapter 4), you should *not* use command abbreviations. An abbreviation that is valid in one revision of the CLI may become invalid if new commands are added. You can avoid this problem by using full command and switch names in your macros. A macro is also easier to understand if it contains complete command names.

Entering Multiple Commands on a Single Line

You can enter two or more separate command lines on a single input line by separating the commands with a semicolon.

For example,

```
) TIME; DATE ⤶
 13:58:29
 22-Sep-91
 )
```

The CLI processes a multiple command line from left to right. If an error occurs, the CLI does not process the commands that follow the one that caused the error. (Chapter 3 provides more information about how the CLI handles an error condition.)

Continuing a Command to the Next Line

Pressing the NEW LINE key normally enters the current command line for execution. But if you type the CLI line continuation character (the ampersand, &) before you press the NEW LINE key, the CLI lets you continue typing the command without trying to execute it.

The continuation character is useful when you need to enter a long or complicated command. You can then split your command line over several input lines. For example,

```
) MOVE/DELETE/V/BEFORE/TLM=16-DEC-91:12:00:00/FTA & ⤶
&):NET:BEETHOVEN:UDD:COMMON:TEST_PROGS MY_FILE1 MY_FILE2 ⤶
```

This command includes several switches and a lengthy pathname argument. Notice that when you continue the command line, the CLI issues this special prompt:

```
&)
```

This prompt indicates that the current input line is part of the previous one.

The continuation character is not a separator. If you break a line where a separator must appear, be sure to type a separator before the ampersand, or at the beginning of the next line.

Do *not* type a separator if you split a command line where you do not want a separator. Notice this variation on the previous command:

```
) MOVE/DELETE/V/BEFORE/TLM=16-DEC-91:12:00:00/FTA :NET:BEE&)
&)THOVEN:UDD:COMMON:TEST_PROGS MY_FILE1 MY_FILE2)
```

The CLI automatically continues the command line if you exceed the maximum line length (256 characters) before pressing the NEW LINE key.

When you continue a line, you can edit only the current input line; there is no way to return to a previous line to make changes or additions. You can discard the entire current command, however, by pressing CTRL−C CTRL−A (as described later in this chapter).

Specifying a Date or Time

Several commands accept switches that require you to specify a date or time. To specify a date, use the format: *dd−mon−yy*. For example, 22–JAN–91.

Notice that you must use an alphabetic abbreviation, not a numeric value, for the month. The abbreviation can be from one through three characters, and must uniquely identify the month. (You can abbreviate October as O, April as AP, and July as JUL, for example.) Separate the parts of the date with a hyphen (−).

To specify a time, use 24−hour notation, in the format: *hh:mm:ss*. For example, 06:30:00 for 6:30 a.m., or 13:30:00 for 1:30 p.m.

The minutes and seconds are optional. The CLI assumes 00 for missing minutes or seconds. Therefore, entering 23 as the time is equivalent to entering 23:00 or 23:00:00. You can also omit a leading 0 from the hours, minutes, and seconds. For example, you can use 9:3 to mean 09:03:00.

To specify a date and a time together, use the format :

dd−mon−yy:hh:mm:ss

For example, for 3:37:18 p.m. on September 14, 1991 you would write

```
14-SEP-91:15:37:18
```

Writing Compact Command Lines with Parentheses and Angle Brackets

Parentheses and angle brackets let you write short command lines that the CLI will expand — saving keystrokes.

Using Parentheses to Repeat Commands

You can use parentheses in a command line to have the command repeat for each item enclosed in the parentheses.

If you enclose a list of arguments, the command executes once for each enclosed argument. The command

```
) WRITE (a b c)↵
```

is equivalent to

```
) WRITE a↵
) WRITE b↵
) WRITE c↵
```

If you place two or more commands within parentheses, the CLI executes each command using the arguments that follow. The command

```
) (TYPE QPRINT) file1↵
```

is equivalent to

```
) TYPE file1↵
) QPRINT file1↵
```

The same principle applies as you build more complex commands. The CLI executes the entire command once for each enclosed item. In the next example, parentheses enclose some but not all of the arguments.

```
) WRITE a (b c) d↵
```

is equivalent to

```
) WRITE a b d↵
) WRITE a c d↵
```

If a command line has two or more lists enclosed in parentheses, the CLI uses the first argument in each list, and then repeats the command with the second argument in each list, and so on until it exhausts the largest group. The following command:

```
) WRITE (a b c) (x y)↵
```

is equivalent to

```
) WRITE a x↵          (The command uses the first item in each group.)
) WRITE b y↵          (Then it repeats, using the second item.)
) WRITE c↵            (Finally, it uses the third item.)
```

Nesting Parentheses

You can include a parenthetical group within another one; this is called *nesting*. When you use this syntax, the CLI treats the innermost list as a unit when it expands the outer list. For example,

```
) WRITE (a (b c) d)↵
```

is equivalent to

```
) WRITE a↵
) WRITE b c↵
) WRITE d↵
```

This alternating treatment continues as you nest parentheses more than two levels deep. The CLI interprets the first-level parentheses as a repeating operator, the second-level parentheses as a grouping operator, the third-level as a repeating operator, the fourth-level as a grouping operator, and so on. You can nest parentheses to any depth.

For example,

```
) WRITE (a (b (c d) e) f)↵
```

is equivalent to

```
) WRITE a↵
) WRITE b c e↵
) WRITE b d e↵
) WRITE f↵
```

Using Angle Brackets to Expand Arguments

Angle brackets are useful if you have two or more arguments that consist of nearly the same characters. You use angle brackets to enclose the character groups that differ, and then attach the bracketed group to the common characters.

The CLI forms arguments by joining each character list within the angle brackets to the characters that appear immediately before the left angle bracket and immediately after the right angle bracket. For example, the CLI expands

```
) QPRINT FILE<1,2,3>↵
```

to

```
QPRINT FILE1 FILE2 FILE3
```

It also expands

```
) QPRINT  FILE<,5,6>↵
```

to

```
QPRINT FILE  FILE5  FILE6
```

The bracketed list can appear anywhere within the string. For example, the CLI interprets the command

```
) QPRINT PROGRESS.<JAN FEB MAR>_91 ↵
```

as

```
QPRINT PROGRESS.JAN_91  PROGRESS.FEB_91  PROGRESS.MAR_91
```

If an argument contains more than one bracketed list, the CLI begins by combining the first element in each group. It continues by cycling through the elements of each group from right to left until all elements in the first group have been used. Observe the next two examples.

```
) WRITE <a b c>.<x y> ↵
 a.x a.y b.x b.y c.x c.y ↵
) WRITE <a b><c d><x y z> ↵
 acx acy acz adx ady adz bcx bcy bcz bdx bdy bdz
```

The total number of arguments equals the product of the items in all groups.

Using Parentheses and Angle Brackets Together

You can use paired parentheses and angle brackets in any combination, nested to any depth. If the command line contains both angle brackets and parentheses, the CLI evaluates the expression in this order:

1. It expands the angle brackets (as explained above) to produce the argument list.
2. It then processes the parentheses, working from left to right.

The following command line includes a parenthetical group and a bracketed group.

```
) WRITE (a b)<c d> ↵
```

The CLI first expands the angle brackets to produce the following argument list:

```
WRITE (ac bc) (ad bd)
```

The CLI then expands the parentheses, resulting in these two commands:

```
WRITE ac ad
WRITE bc bd
```

In the next example,

```
) WRITE (<a b><c d>) ↵
```

the CLI begins by expanding the two pairs of angle brackets. The result is

```
WRITE (ac ad bc bd)
```

After processing the parentheses, the CLI generates the following commands:

```
WRITE ac
WRITE ad
WRITE bc
WRITE bd
```

Using Square Brackets to Specify the Contents of a File

When you type a filename and surround it with square brackets, the CLI substitutes the contents of the file for the bracketed filename. You can use this feature to store arguments in files — to help reduce keystrokes and prevent typing errors. For example, assume you want to print the following files in order:

```
OUTLINE.DOC SYNOPSIS.DOC INTRODUCTION.DOC BODY.DOC
SUMMARY.DOC
```

You could type all the filenames manually after the QPRINT command, at the cost of some effort. You could use a template (wildcard) character — for example QPRINT +.DOC — to print the files, but this would probably not print them in the order you want. Nor would alphabetical order, as with QPRINT/SORT +.DOC, produce the sequence you want.

But if you include the filenames in a disk file (using a text editor or the CLI CREATE/I command), you can print them easily in the order you want. The disk file, which you could call PROJECT.FILES, might contain these lines:

```
OUTLINE.DOC &
SYNOPSIS.DOC &
INTRODUCTION.DOC &
BODY.DOC &
SUMMARY.DOC &
```

The ampersand continues the text to the next line; without it the NEW LINE character after OUTLINE.DOC would act as a command delimiter.

You can print all the files in the correct order by typing

```
) QPRINT [PROJECT.FILES]↵
```

The CLI substitutes the filenames in PROJECT.FILES for the filename, producing the command

```
QPRINT OUTLINE.DOC SYNOPSIS.DOC INTRODUCTION.DOC BODY.DOC
SUMMARY.DOC
```

Probably you would want to test the indirect file first, using the TYPE command instead of QPRINT. Or you could use the WRITE command to preview the results; as with WRITE [PROJECT.FILES].

This technique, using a file of filenames, works with any CLI command that accepts arguments. It is most useful when you want to ensure that the command processes the files in a specific order.

Previewing CLI Commands

Placing WRITE ahead of a command results in the CLI's displaying the command exactly the way it would execute it if WRITE were not present. If the display is not what you want, simply change the command and try again. Study the following dialog. If you want, duplicate it at your terminal and make any changes you want to the commands while observing what happens. Although the dialog previews the effects of parentheses and angle brackets in TYPE commands, you can place WRITE ahead of *any* CLI command to see exactly what CLI command would execute if WRITE were not present.

```
) WRITE TYPE (FILEA FILEB)↵
 TYPE FILEA
 TYPE FILEB

) WRITE  TYPE  FILE<A,B>↵
 TYPE FILEA FILEB

) WRITE  TYPE  (FILEA FILE<B,C>)↵
 TYPE FILEA
 TYPE FILEB
 TYPE FILEC

) WRITE TYPE FILE<A B>(C D E)↵
 TYPE FILEAC FILEBC
 TYPE FILEAD FILEBD
 TYPE FILEAE FILEBE

) WRITE  TYPE  FILE(A B)<C D E>↵
 TYPE FILEAC FILEAD FILEAE
 TYPE FILEBC FILEBD FILEBE

) WRITE  TYPE  FILE(<A B><C D>)↵
 TYPE FILEAC
 TYPE FILEAD
 TYPE FILEBC
 TYPE FILEBD

) WRITE  TYPE  FILE(<A B <C D>)↵
 Error: Mismatched bracket types
 WRITE,TYPE,FILE(<A,B,<C,D>)
```

If you restore the missing > symbol you will get the same results as from the previous command. In the next command there is no space between the) and (.

```
) WRITE  TYPE  FILE<(A B)(C D)>↵
 TYPE FILEAC
 TYPE FILEBD
```

In the next command there is one space between the) and (.

```
) WRITE  TYPE  FILE<(A B) (C D)>↵
```

TYPE FILEA FILEC
TYPE FILEB FILED

```
) WRITE (TYPE QPRINT) FILE<A BC D>↵
```

TYPE FILEA FILEBC FILED
QPRINT FILEA FILEBC FILED

Reviewing Command History (CLI32 Only)

The CLI32 program includes a HISTORY command and features that let you recall and edit commands you have previously typed. The default number of commands stored is 25, but you can specify more or fewer commands.

To display the commands, use the HISTORY command. You can redisplay, and then use screen edit control characters to edit, previously typed commands using the uparrow (↑) and downarrow (↓) keys. You can redisplay commands at any point, even while typing another command. For example,

) HISTORY↵	(Type HISTORY without an argument; CLI displays commands previously typed)
1 WHO	
2 ACL MARK_II	
3 SED MYFILE ↑	(Press uparrow key)
) ACL/V MARK_II ↑	(CLI displays command previously typed; press uparrow again)
) ACL/V MYFILE↵	(CLI displays command previously typed; press NEW LINE to execute it)
JOAN,OWARE +,E	(CLI executes the ACL command)

History features are further described in Chapter 5, under HISTORY.

How the CLI Processes Command Lines

When you type one or more characters and press NEW LINE, the CLI processes your line of text (command line) as follows.

1 First, the CLI verifies that any parentheses and brackets in the command are properly matched. If any are not, the CLI displays an error message and stops executing the command line.

2 The CLI verifies that the characters used in the first unbroken text string are legal characters. If there are any illegal characters, the CLI displays an error message and stops executing the command line.

3 The CLI compares the leading text string to its command table. If the text string matches a command, the CLI assumes you typed a command. It checks for universal command switches (described at the beginning of Chapter 5) and then checks any other switches and arguments in the context of that command. If it finds any illegal arguments, characters, or invalid switches, it displays an error message and stops executing the command line.

   If there are no syntax errors, the CLI tries to execute the command. It processes any errors during execution according to the severity of the error and the CLI exception (error class) setting.

4 If the leading text string has no illegal characters but is not a CLI command, the CLI tries to execute it as a macro. The CLI searches for a file whose name matches the leading text string (up to a slash, /, which indicates a switch), with the .CLI suffix; then if that search fails, the CLI searches for the name without the .CLI suffix. For example, if you type XXX/YY ZZZ, the CLI searches for the filename XXX.CLI, then for XXX.

   If the CLI finds such a file, it assumes a CLI macro and tries to execute the contents of the file as if they were CLI commands. It processes any errors during execution according to the severity of the error and the CLI exception setting.

   If the CLI cannot find a matching filename, it searches for a program file to execute (CLI32) as follows or gives up and displays *Not a command or macro* (CLI16).

5 When CLI32 cannot recognize what you typed as a command, or find it as a macro, it searches for a program file (filename with .PR suffix). If the CLI can find the filename with .PR suffix, it tries to execute the file as a program. For example, if you type XXX/YY ZZZ, the CLI searches for the filename XXX.CLI, or XXX to execute as a macro, and then searches for XXX.PR to execute as a program. It processes any errors during execution according to the program's error handling routines (if any).

   If CLI32 cannot find a matching filename with the .PR suffix, it gives up and displays *Not a command, macro, or program.*

Positioning the Screen Cursor

The earlier section "Screenedit Control Characters" explains how to move the cursor within a line. Other special characters in CLI commands allow you to move the cursor anywhere on your screen. For example, the following command will move the cursor to the 25th column and then to the 15th line (row) on your screen; from there it displays *At row 15 and column 25.*

```
) WRITE [!ASCII 220, 230, 216]At row 15 and column 25⤶
```

Commands such as the previous one, when executed from within a macro, help to display menus for yourself or for other users on your system. Inexperienced users can then log on, type a macro name, and have a list of choices.

The explanation of the WRITE command in Chapter 5 lists the special numbers that position the cursor anywhere on a terminal screen. This explanation also includes a macro that displays a menu and receives a user's response.

Locking the CLI

Sometimes, you may want to disable certain commands in the CLI — perhaps so that you can leave your terminal unattended without the risk of having files deleted or private files read. Or, as a system manager, you may want to lock the CLI running on the system console. That CLI, the master CLI, is a privileged process and — if the system console is left unattended in a public place — represents a security risk.

Both CLIs (CLI32 and CLI16) offer locking features that let you disable certain commands. With CLI32, you can easily create a password and lock your own CLI. With CLI16, creating a password is fairly complex, and you cannot lock your own CLI; you must execute a different CLI, :LOCK_CLI.PR.

Locking CLI32

To lock CLI32, you must first define a password using the PASSWORD command. This password persists only as long as your CLI process lasts. However, you can store the password in a file with the PASSWORD/WRITE= command, after which you can re-establish it for a new CLI process with the PASSWORD/READ= command. Then you can retrieve the password with the LOCK/FILE= command to lock the CLI without having to type the password. (Alternatively, you can use the command PASSWORD/NOPROMPT/READ=filename, which also lets you lock the CLI without typing the password.) You can set the password and lock the CLI automatically from your log-on macro (or :UP.CLI, for the system console) using the commands

```
PASSWORD/READ=password-pathname
LOCK/FILE=password-pathname
```

After setting a password, you can issue the LOCK command to lock any or all CLI commands. The format of the LOCK command is

```
LOCK      [CLI–command–to–disable] [...]
LOCK/CX   [EXEC–command–to–disable] [...]
```

If you include one or more arguments, the CLI will disable those commands only. If you omit arguments, the CLI disables all CLI commands that relate to security, including the following:

BLOCK	DUMP	PRIVILEGE	SUPERPROCESS
BYE	EXECUTE	PROCESS	SUPERUSER
CHAIN	INITIALIZE	QBATCH	TERMINATE
CONNECT	JPINITIALIZE	QFTA	XEQ
COPY	JPRELEASE	QPLOT	
DEBUG	LOAD	QSUBMIT	
DELETE	MOVE	RELEASE	

Thus, you cannot execute programs from a CLI that was locked via LOCK without an argument.

When you lock it, the CLI automatically turns off Superuser, Superprocess, and/or System Manager privilege if any one was turned on. The process termination sequences CTRL–C CTRL–B, CTRL–C CTRL–E, and CTRL–D CTRL–D are ignored.

If you include the /CX switch without arguments, the CLI disables all EXEC commands. EXEC command issues are further explained in Chapter 5, the LOCK command (CLI32 version) and in *Managing AOS/VS and AOS/VS II.*

To unlock a locked CLI, use the UNLOCK command.

NOTE: If you forget the password, you cannot exit from this CLI. You will need to terminate it (or have it terminated) from a superior process.

Locking CLI32 — Example

```
) LOCK↵
Warning: No password is in effect

) PASSWORD↵
Password: JOAN↵                 (Password does not echo.)
Confirm password: joan↵
) LOCK↵
Password: JOAN↵
) XEQ  MYPROG↵
Error: Command is locked, XEQ

) PASSWORD/WRITE=:UDD:JOAN:PW_FILE↵
) UNLOCK↵
Password:  JOAN↵                (Password does not echo.)
) XEQ  MYPROG↵
... (MYPROG executes) ...
```

In this command sequence, a person tries to lock the CLI, fails because no password has been defined, defines a password, locks the CLI, tests a locked command, and finally writes the password to file PW_FILE (encrypted) from which she can have the CLI re-establish it and lock the CLI later via the commands

```
PASSWORD/READ=:UDD:JOAN:PW_FILE
LOCK/FILE=:UDD:JOAN:PW_FILE
```

Ultimately this person unlocks the CLI and executes the program.

Locking CLI16

There is no command to lock the standard CLI16 process. If you want to lock a 16-bit CLI, you must execute the program :LOCK_CLI. LOCK_CLI is a special, lockable CLI that requires a password to unlock or terminate. It's designed to safeguard the system console from unauthorized people. In the unlocked state, LOCK_CLI is identical to the standard CLI, except that it accepts the command LOCK. The LOCK command locks this CLI, preventing it from executing security-related commands. While locked, it ignores the same commands CLI32 ignores (except those that don't exist in CLI16). For more information on LOCK_CLI, see the LOCK command, CLI16 version, in Chapter 5.

CLI32 No-Interrupt (/NOCA) Switch

As system manager, you can allow a user access to some system facilities, yet prevent access to the CLI. You can accomplish this through the command

EXECUTE CLI32/NOCA *program–name*

or through an IPC file that uses this feature. You can set up the user's AOS/VS II profile and IPC to execute CLI32 with the /NOCA switch. The argument *program–name* is the name of a macro or program that provides access to system functions that you have preselected. The CLI32 switch blocks console interrupts (key sequences such as CTRL–C, CTRL–A) as long as the macro or program is running.

The following example shows an initial macro calling a menu macro that allows user *guest* to request specified tasks, but not to access the CLI command line. Assume that the guest profile specifies an IPC file :UDD:GUEST:LOGON.CLI, which executes CLI32/NOCA MENU.CLI. The menu macro contains an infinite loop that prompts for a menu selection and calls the associated program or macro. The option to end the session exits the loop and executes the BYE command.

CLI32/NOCA Example: LOGON.CLI Macro

```
\\ LOGON.CLII                              Calls MENU.CLI
\\
SEARCHLIST :UDD:GUEST :UTIL :
XEQ CLI32/NOCA :UDD:GUEST:MENU.CLI
BYE
```

CLI32/NOCA Example: MENU.CLI Macro

```
\\ MENU.CLI                                  Presents list of available options
\\
CLASS1 IGNORE; CLASS2 IGNORE                 \\ Ignores errors from incorrect
                                             \\ option input
CHARACTERISTICS/OFF/ST                       \\ Turns off tab simulation: when on, ST
                                             \\ causes WRITE comands specifying
                                             \\ row 9 or col 9 to position incorrectly


[!LOOPSTART]                                 \\ Starts option scan loop

STRING/NAME=1   CEO                          \\ Calls CEO
STRING/NAME=2   Console                      \\ Calls macro to get console information
STRING/NAME=3   Queues                       \\ Calls macro to get queue information


WR/NONEWLINE [!ASCII 214 220 235 210]        \\ Erases screen, moves near center


WRITE MENU                                   \\ Each command positions cursor for
                                             \\ number and for description


WR [!ASCII 220 224 214] OPTION [!ASCII 220 240 214] DESCRIPTION \\ Writes heads
WR [!ASCII 220 227 215] 1 [!ASCII 220 234 215] Invokes CEO       \\ Writes options
WR [!ASCII 220 227 216] 2 [!ASCII 220 234 216] Displays your console address
WR [!ASCII 220 227 217] 3 [!ASCII 220 234 217] Displays system queues
WR [!ASCII 220 227 220] 4 [!ASCII 220 234 220] Ends your session  \\ Writes options
WR/NONEWLINE [!ASCII 220 204 222]            \\ Positions prompt message on screen


STRING/NAME=option [!READ/LENGTH=1 Type option number: ]        \\ Reads choice


[!EQUAL,[!STRING/NAME=option],4]             \\ Tests for end session option
   WRITE
   CHARACTERISTICS/OFF/ST                    \\ Turns  tab simulation back on
   [!EXIT/LOOP]                              \\ Returns to command following loop
[!END]                                       \\ Marks end of end session conditional
```

```
                                        \\ Tests for a valid option
VAR0 0                                  \\ Sets option test results to zero
[!NEQUAL,[!STRING/NAME=option],1]       \\ True if option input not 1
   VAR0 [!UADD [!VAR0] 1]               \\ Saves result
[!END]                                  \\ End of option 1 conditional
[!NEQUAL,[!STRING/NAME=option],2]       \\ True if option input not 2
   var0 [!uadd [!var0] 1]               \\ Adds result
[!END]                                  \\ End option 2 conditional
[!NEQUAL,[!STRING/NAME=option],3]       \\ True if option input not 3
   var0 [!uadd [!var0] 1]               \\ Adds result
[!END]                                  \\ End option 3 conditional
[!nequal,[!var0],3]                     \\ True if option is 1, 2, or 3
   [!STRING/NAME=[!STRING/NAME=option]]\\ Calls selected macro
                                        \\ or program
[!END]                                  \\ Marks end of option test conditional
[!LOOPEND]                              \\ Marks end of option scan loop

BYE                                     \\ Ends session
```

CLI32/NOCA Example: Macros called from MENU.CLI

```
\\ CONSOLE.CLI                          Executes CLI commands for Option 2
\\
WRITE [!ASCII 214]
WRITE
CONINFO                                 \\ Obtains terminal address informaton
PAUSE 5


\\ QUEUES.CLI                           Executes CLI commands for Option 3
\\
WRITE [!ASCII 214]
WRITE Press CTRL–Q to advance to next screen.
WRITE
CHARACTERISTICS/PM                      \\ Sets screen to page mode
QDISPLAY/VERBOSE                        \\ Displays queues
PAUSE 5
CHARACTERISTICS/OFF/PM                  \\ Sets  page mode off
```

A user who logs on as guest sees the following menu.

```
                    MENU

          OPTION      DESCRIPTION

             1     Invokes CEO
             2     Displays your console address
             3     Displays system queues
             4     Ends your session

Type option number: _
```

Each task option the user selects returns to the menu upon completion. Pressing CTRL–C, CTRL–A has no effect as long as the macro runs. Choosing option 4 logs the user off the system. The macro detects and ignores input other than 0 through 4. Class 1 and 2 exceptions are ignored so that a user who accidentally or intentionally presses a special character or function key cannot cause the macro to terminate and expose the CLI interface. Assuming that the AOS/VS II profile is properly set up and the CEO profile does not allow use of the CLI, the guest account lets the user perform predetermined tasks, but provides no direct access to CLI commands.

Refer to the WRITE command description in Chapter 5 for a more detailed menu example.

End of Chapter

Format of a Pathname

As mentioned earlier, a pathname tells the system where to locate a file. Starting in the directory specified by the prefix (or the working directory if there is no prefix), the pathname lists the directories that you must go through (if any), and ends with the name of the target file. A colon separates each name within the pathname. In this way you can specify any one file within the entire system directory tree.

The general format for a pathname is

*[prefix][directory–name:][...]*filename

The filename can, of course, belong to a directory file.

NOTE: Do not confuse the root directory name (:) with the colon used to separate filenames in a pathname. A colon at the beginning of a pathname *always* represents the root directory, while a colon within a pathname simply separates filenames.

Except for the caret prefix (^) and =, pathnames always move down the directory tree. If you have two filenames in a pathname, the second one must be immediately subordinate to the first.

A file's full pathname begins at the root directory (:) and continues down through the directory tree to that file. Because it begins at the root directory, a full pathname allows the system to locate the file regardless of your current directory.

The prefixes = and ^, however, let you refer to a file by specifying a path that is relative to your working directory. In other words, such a pathname depends on your current location within the directory tree. If you were to change your working directory, you would have to change the pathname to refer to the same file.

Using a pathname lets you refer to any file within the directory tree without changing your working directory.

Figure 2–2 shows a typical directory tree with four user directories and several data files. (In the figure, ovals and circles represent directories and rectangles represent nondirectory files.)

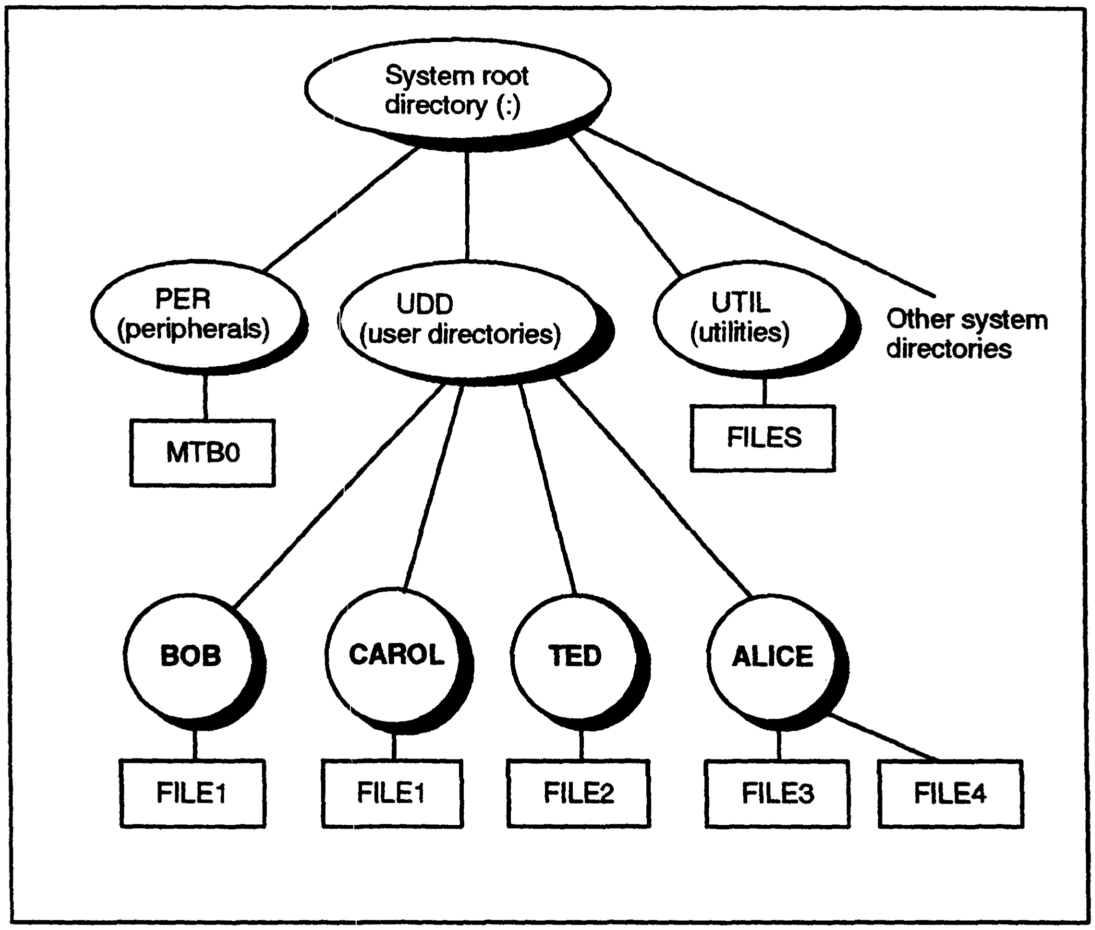


Figure 2–2 A Directory Tree with User Files

In Figure 2–2, assume that your working directory is CAROL. The pathname is

:UDD:CAROL

That is, beginning at the root directory (:), the path goes to the UDD directory and then to the CAROL directory.

If you refer to FILE1, the system uses the file within your working directory. You could also refer to this file as =FILE1 (using the prefix that begins at your working directory) or the complete pathname :UDD:CAROL:FILE1.

There is no direct path from directory CAROL to FILE2 in directory TED. If you issue the command

```
) TYPE FILE2⤶
```

the system responds with an error message that says the file does not exist. You must use a pathname to enable the system to locate the file you want. For example

```
) TYPE :UDD:TED:FILE2⤶
```

Table 2–3 lists several pathnames that you could use to refer to files in Figure 2–2 (assuming your working directory is :UDD:CAROL).

CLI Prompt

When it is ready for your input, the CLI displays the prompt

```
)
```

You can associate as many as eight CLI commands with the prompt. Whenever it displays the prompt and its trailing blank space, the CLI first executes the commands (if any) associated with the prompt. (If you want to change the prompt character itself, use the PREFIX command described later in this chapter and in Chapter 5.) You can use the PROMPT command to display the current prompt commands, or to associate CLI commands with the prompt. The commands you associate with the prompt cannot have arguments or switches.

One common use of the prompt commands is to display the name of the working directory with each prompt. You will find this helpful if you frequently change directories. If you want the system to display the time with each prompt, you can define the TIME command as a prompt command.

The following dialog shows how the PROMPT command works.

```
) PROMPT↵
)
```

The CLI has displayed a blank line. Now change the prompt.

```
) PROMPT TIME DIRECTORY↵

 17:08:22
 :UDD1:LISA

) WRITE The date is; DATE↵

The date is
19-Jan-91
17:08:46
:UDD1:LISA
```

Display the current prompt.

```
) PROMPT↵

 TIME DIRECTORY
 17:08:56
 :UDD1:LISA
```

Kill the new prompt, restoring the default.

```
) PROMPT/K↵
)
```

Super Privilege Modes

The super privilege modes are Superuser, Superprocess, and System Manager. You can activate any of these only if your user profile grants the privilege.

Superuser Mode

The SUPERUSER command lets you display whether Superuser mode is on or off. If your user profile grants you Superuser privilege, you can turn Superuser mode on or off.

When you turn Superuser mode on, you have access to the entire filing system. If you do not have the Superuser privilege or if you have Superuser mode turned off, you are subject to the restrictions imposed by access control lists. The Superuser prompt is shown in Table 3–2.

Superprocess Mode

The SUPERPROCESS command lets you display whether Superprocess mode is on or off. If your user profile grants you Superprocess privilege, you can turn Superprocess mode on or off.

When you have Superprocess mode turned on, you can terminate, change the priority of, block, or otherwise influence any process on the system. If you do not have the Superprocess privilege, or if you do not have Superprocess mode turned on, you can affect only the current CLI process and its sons. The Superprocess prompt is shown in Table 3–2.

System Manager Mode

The PRIVILEGE SYSTEMMANAGER command lets you display whether System Manager mode is on or off. If your user profile grants System Manager privilege, you can use the command to turn the privilege on or off.

When you have System Manager mode turned on, you can issue EXEC commands, set the system time or date, and other operations explained in *Managing AOS/VS and AOS/VS II.* The System Manager prompt is shown in Table 3–2, next.

Super Privilege Mode Prompts

When one or more super privilege modes are on, the CLI displays a prompt prefix that indicates the active modes. CLI16 and CLI32 use the same prompt prefixes, as shown in Table 3–2.

Table 3–2 Super Privilege Mode Prompt Prefixes

Mode(s) Activated	**Prefix**
Superuser	*Su)*
Superprocess	*Sp)*
System Manager	*Sm)*
Superuser and Superprocess	*SuSp)*
Superuser and System Manager	*SmSu)*
Superprocess and System Manager	*SpSu)*
Superuser, Superprocess, and System Manager	*SmSpSu)*

List File

The current list file setting determines which file receives output from a CLI command or program that writes to the generic @LIST file.

A program can be written to send messages to the generic file @LIST. Before running such a program, you specify which file is to be the current list file. Program output then goes to that file. Later, you can run the program again, but with a different list file. Thus, without changing the program's code, you can direct program output to different files.

All CLI commands accept the /L switch, which in interactive mode causes the CLI to write output from the command to the current list file rather than to the terminal. In batch mode, the /L switch causes the CLI to place output from the command into the current list file rather than into the batch output file. So if you want to direct output from a command to a specific file, you can designate a list file, and then execute the command with the /L switch.

You can also direct command output to a specific file by using the switch

/L=filename

If the file does not exist, the CLI automatically creates it for you. If the file already exists, the CLI appends the output to the end of that file.

You can use the LISTFILE command to display the current list file or to set it. The CLI also provides the pseudomacro !LISTFILE, which represents the pathname of the current list file.

For example, the following command invokes the SCOM utility program to compare files FILE1 and FILE2, and sends the output from the program to the console.

```
) XEQ SCOM/L=@CONSOLE FILE1 FILE2)
```

To send output to the line printer, you would replace /L=@CONSOLE with /L=@LPT; or you could specify a disk file, say FILE.DIFFS, by using /L=FILE.DIFFS.

The following command creates a sorted list of filenames and sends it to file TEMP_0423. If file TEMP_0423 does not exist, the CLI creates it and writes into it. If the file already exists, the CLI writes into it at the end of the file.

```
) FILESTATUS/SORT/L=TEMP_0423)
```

Data File

The current data file setting determines the file that a program uses when it reads from the generic @DATA file.

A program can be written to read from the generic file @DATA. Before running such a program, you specify which file is to be the current data file. The program will then read from that file. Later, you can run the program again, but with a different data file. Thus, without changing the program's code, you can run the program using different data files.

You can use the DATAFILE command to display the current data file or to set it. The CLI also provides the pseudomacro !DATAFILE, which represents the pathname of the current data file.

User Log File

The current user log file setting determines the file that the CLI uses to record your CLI commands and the responses they receive. (The log file does not include output from the TYPE command, nor does it record any output from another program.)

You can use the LOGFILE command to display the name of your current log file or to set it.

CLI Variables

The CLI includes 10 built-in variables, which you can use to store and retrieve values. Each of these variables, named VAR0 through VAR9, can hold a double-precision value (ranging from 0 through 4,294,967,295). The initial value of a CLI variable is 0.

You can display the value of a variable or set it by using the command VARn (where n is from 0 through 9). The CLI also provides the pseudomacros !VAR0 through !VAR9, which represent the current value of the corresponding CLI variable. You can use numeric operators such as !UADD and !UMULTIPLY with these for integer computations. For example

```
) VAR0 2 ↵                            (Set the variable VAR0 to 2.)
) WRITE [!VAR0] ↵                     (Display the value of VAR0.)

 2

) VAR0 [!UADD [!VAR0, 2]] ↵           (Set the value to itself plus 2.)
) WRITE  [!VAR0] ↵                    (Display the value.)

 4
```

CLI32 (but not CLI16) has a variable called VAR that allows named variables. You access it via the form VAR/NAME=xxx. Like the numeric variables, it holds integer values only. For example

```
) VAR/NAME=COUNTER 2 ↵                (Set the variable COUNTER to 2.)
) WRITE [!VAR/NAME=COUNTER] ↵         (Display the value of COUNTER.)

 2
```

CLI String(s)

The CLI maintains a 127-character String, which you can use (normally via a macro) to store and retrieve text.

For example, when executing a program, you can use the /S switch to direct the program's termination message to the CLI String. A macro or batch job can then retrieve and process the termination message.

Macros often use the CLI String to store the response that a user types to a macro prompt. The macro can then test the contents of the CLI String to determine what action to take.

You can use the STRING command to display the contents of the CLI String, or to assign a text string to it. The CLI also provides the pseudomacro !STRING, which represents the current contents of the CLI String.

CLI32 (but not CLI16) provides named strings. You can access these with the /STR switch in CLI32 commands and via the STRING command and !STRING pseudomacro with the /NAME switch. Like named variables, named strings provide a significant advantage for CLI32 over CLI16. For example

```
) SPACE/STR=DISK_SPACE ↵                  (Place the result of the command in a
                                           string named DISK_SPACE.)

) WRITE [!STRING/NAME=DISK_SPACE] ↵       (Display the value of DISK_SPACE.)
 Max 20000   Cur 13362   Rem 6638
```

Device Characteristics

Your terminal's characteristics determine how the terminal interprets input and sends output. There are many individual settings that make up your terminal's characteristics, including the terminal type, the number of characters per line, parity, and baud rate.

You can use the CHARACTERISTICS command to display a device's current characteristics, or to change them. The description of the CHARACTERISTICS command (in Chapter 5) lists the switches that apply for individual settings. This information will help you interpret the display of the current settings, which is given in terms of these switches.

Using a Conditional Pseudomacro as an Argument

You can use a conditional pseudomacro as an argument to another conditional pseudomacro. This means that the first and second arguments can vary depending on specified conditions.

The macro TEST3.CLI contains the following lines:

```
COMMENT This is macro TEST3.CLI.
[!EQUAL,&
            [!EQUAL,%1%,%2%]yes[!ELSE]1<>2[!END],&
            [!EQUAL,%2%,%3%]yes[!ELSE]2<>3[!END]]
            WRITE All three arguments are equal.
[!ELSE]
            WRITE The arguments are not all equal.
[!END]
```

This macro tests whether the first, second, and third arguments are all equal. The second and third lines are conditional pseudomacros that are on one line instead of three. These lines are easily read, so there is no need to have them span three lines each.

The first line contains the conditional pseudomacro !EQUAL, whose arguments are the following conditional pseudomacros:

```
[!EQUAL,%1%,%2%]yes[!ELSE]1<>2[!END]
```

and

```
[!EQUAL,%2%,%3%]yes[!ELSE]2<>3[!END]
```

For the first statement to be true, both of the conditional pseudomacros must also be true. Table 4–4 illustrates the possibilities.

Table 4–4 Possible Conditions for the Macro TEST3.CLI.

Condition	Resolution	Result
%1% = %2% and %2% <> %3%	[!EQUAL,yes,23]	False
%1% = %2% and %2% = %3%	[!EQUAL,yes,yes]	True
%1% <> %2% and %2% = %3%	[!EQUAL,12,yes]	False
%1% <> %2% and %2% <> %3%	[!EQUAL,12,23]	False

Using Parentheses to Unify a String Argument

When using !EQUAL or !UNEQUAL, you must surround a string argument with parentheses if the string contains a separator character (such as a space or comma). If you surround the string with parentheses, the CLI treats the entire string as a single argument. Otherwise, the CLI assumes that the separator marks the end of the string.

Suppose, for example, that you want to know if the CLI String contains the error message INPUT FILE ERROR. The statement

```
[!EQUAL,[!STRING],INPUT FILE ERROR]
```

will cause an error, because the CLI sees INPUT, FILE, and ERROR as separate arguments; the !EQUAL pseudomacro accepts two arguments only.

Your intention in this case is to treat INPUT FILE ERROR as a single argument, and compare that string with the contents of the CLI String. Parentheses enable you to do this:

```
[!EQUAL,([!STRING]),(INPUT FILE ERROR)]
```

Note that you must surround [!STRING] with parentheses as well because its contents may comprise multiple words. If you enclose only one argument with parentheses, the comparison will always be unequal.

Using Loop Pseudomacros

CLI32 provides pseudomacros that you can use to execute a series of CLI commands within a loop. You can set up the loop to execute indefinitely, repeat a number of times that you specify, or repeat until a condition is recognized. This loop structure executes in less time and uses less memory than a recursive macro.

Every loop begins a series of statements with !LOOPSTART and ends with !LOOPEND. The series can include !EXIT, !EXIT/LOOP, or !EXIT/MACRO pseudomacros. The basic format follows.

```
[!LOOPSTART [iteration_count] ]  (Required to mark start of loop; count optional)
. . .                            (Optional location for commands)
  [!EQUAL,arg1,arg2]             (Optional test for exit condition)
    [!EXIT/LOOP]                 (Alternative method for leaving loop)
  [!END]                         (Required delimiter for conditional pseudomacro)
. . .                            (Optional location for commands)
[!LOOPEND]                       (Required to mark end of loop)
```

Loop execution ends if

- An iteration count is specified and the count is satisfied
- A conditional pseudomacro detects an exit condition
- You type a CTRL–C, CTRL–A sequence

Refer to Chapter 5 for detailed description of each the loop pseudomacros.

Loops can use pseudomacros within dummy arguments to simulate the passing of arguments in recursive macro calls. The following macro named SIM_RECURSE.CLI uses a dummy argument in a loop to simulate a recursive macro passing %2–% on each call.

```
COMMENT This is macro SIM_RECURSE.CLI.
VAR/NAME=ARG    1
[!LOOPSTART]
  [!EQUAL,%[!VAR/NAME=ARG]%,]
    [!EXIT/LOOP]
  [!END]
WRITE Arguments  [!VAR/NAME=ARG] to last – %[!VAR/NAME=ARG]–%
VAR/NAME=ARG     [!UADD,[!VAR/NAME=ARG],1]
[!LOOPEND]
```

```
) SIM_RECURSE [!FILENAMES/NOEQUAL/SORT] ⤶
Arguments 1 to last – CLI.DL CLI.DS CLI.MAP.LS CLI.PR CLI.ST
Arguments 2 to last – CLI.DS CLI.MAP.LS CLI.PR CLI.ST
Arguments 3 to last – CLI.MAP.LS CLI.PR CLI.ST
Arguments 4 to last – CLI.PR CLI.ST
Arguments 5 to last – CLI.ST
```

Macro Examples: the Calculator and Space Percentage Macros

This section includes two sample macros: CALC.CLI, which does integer calculations, and SPACEX, which derives the percentage of space usage on system disk units.

Macro CALC.CLI

Macro CALC.CLI performs the following arithmetic operations:

- Adds two integers
- Subtracts an integer from another
- Multiplies two integers
- Divides one integer by another
- Converts a decimal number to an octal number
- Convert an octal number to a decimal number

You use a switch to indicate the operation you want to perform on the argument(s). Table 4–5 shows the syntax for the macro CALC and gives examples. Figure 4–1 shows the macro itself.

The following switches let you select files by date and time.

/AFTER/TCR=	(CLI32 only) Selects files *created after* the specified date, time today, or date and time. The format for the date is dd–mon–yy, for time is hh:mm:ss (on a 24–hour clock), and for date and time together is dd–mon–yy:hh:mm:ss. An example for October 4, 1990 at 1:04:46 p.m.: AFTER/TCR=04–OCT–90:13:04:46
/AFTER/TLA=	Selects files *last accessed after* the specified date, time today, or date and time. The format for the date is dd–mon–yy, for time is hh:mm:ss (on a 24–hour clock), and for date and time together is dd–mon–yy:hh:mm:ss. An example for October 4, 1990 at 1:04:46 p.m.: /AFTER/TLA=04–OCT–90:13:04:46
/AFTER/TLM=	Selects files *last modified after* the specified date, time of day, or date and time. The format for the date is dd–mon–yy, for time is hh:mm:ss (on a 24–hour clock), and for date and time together is dd–mon–yy:hh:mm:ss. An example for October 4, 1990 at 1:04:46 p.m.: /AFTER/TLM=04–OCT–90:13:04:46
/BEFORE/TCR=	(CLI32 only) Selects files *created before* the specified date, time of day, or date and time. The format for the date is dd–mon–yy, for time is hh:mm:ss (on a 24–hour clock), and for date and time together is dd–mon–yy:hh:mm:ss. An example for October 4, 1990 at 1:04:46 p.m.: /BEFORE/TCR=04–OCT–90:13:04:46
/BEFORE/TLA=	Selects files *last accessed before* the specified date, time of day, or date and time. The format for the date is dd–mon–yy, for time is hh:mm:ss (on a 24–hour clock), and for date and time together is dd–mon–yy:hh:mm:ss. An example for October 4, 1990 at 1:04:46 p.m.: /BEFORE/TLA=04–OCT–90:13:04:46
/BEFORE/TLM=	Selects files *last modified before* the specified date, time of day, or date and time. The format for the date is dd–mon–yy, for time is hh:mm:ss (on a 24–hour clock), and for date and time together is dd–mon–yy:hh:mm:ss. An example for October 4, 1990 at 1:04:46 p.m.: /BEFORE/TLM=04–OCT–90:13:04:46.

The minutes and seconds are optional when specifying the time. You can use a combination of /BEFORE and /AFTER switches to specify a range, provided that both switches refer to the time created (/TCR switch, in CLI32 only), the time last accessed (/TLA), or to the time last modified (/TLM). For example, you cannot use /TLA and /TLM in the same command. An example of a combination of /BEFORE and /AFTER switches is in the FILESTATUS command earlier in this section.

Universal Command Switches

The command descriptions in this chapter refer to this section to explain the following universal switches: /1, /2, /L, /Q, /STR=, and /ESTR=. Unlike other command and switch names, /STR= and /ESTR= cannot be abbreviated. All CLI32 *commands,* except those noted (but not necessarily pseudomacros and utilities), accept all these switches. CLI16 commands accept all but the /STR= and /ESTR= switches. Most utilities offer the /L switch. The universal command switches function as follows:

/1={IGNORE }
{WARNING}
{ERROR }
{ABORT }
	Sets the Class 1 exception response to the specified severity level for this command only. The /1= switch lets you override the current Class 1 setting for the duration of this command.

/2={IGNORE }
{WARNING}
{ERROR }
{ABORT }
	Sets the Class 2 exception response to the specified severity level for this command only. The /2= switch lets you override the current Class 1 setting for the duration of this command.

/ESTR=string	(CLI32 only). Writes into the named string any error message that a CLI32 command or user program generates. Any STRING command or !STRING pseudomacro can then access that string using the /NAME= switch. If no error has occurred, the named string returns a null. Executing a command that includes this switch deletes any content that the named string may have had previously. For user programs, the named string stores the message string that the program returns. For example, a C program would store the "program 'exit'ed with status of n" string.
/L	Writes the CLI output to the current list file (LISTFILE command) instead of to @OUTPUT or to the batch output file.
/L=pathname	Writes the CLI output to the specified file instead of to @OUTPUT or to the batch output file. To send output to the default line printer, use @LPT as a pathname.
/Q	Turns Squeeze mode on (compresses output, overrides the Squeeze off setting) for the duration of this command. It has no affect on output from the TYPE command.
/STR=string	(CLI32 only). Writes command output (but not error messages) into the named string in normalized form (with arguments separated by commas). Any STRING command or !STRING pseudomacro can then access that string using the /NAME= switch. The /STR switch is useful only with CLI commands that can produce output, such as ACL and TIME. If a command with the /STR switch produces no output, the CLI sets the named string to null. The switch does not work as usual — it returns null — with the COPY and TYPE commands.

The following example shows how /STR works.

```
) SPACE⤶                                (Display disk usage numbers.)
Max 20000, Cur 13600, Rem 6400

) SPACE/STR=SP.STRING⤶                  (Get disk usage numbers and
                                        save them in SP.STRING.)

) STRING/NAME=SP.STRING⤶                (Display SP.STRING contents.)
Max,20000,Cur,13600,Rem,6400

) DIRECTORY/I/STR=SP.STRING⤶            (This command produces no
                                        output, nullifying the string.)

) STRING/NAME=SP.STRING⤶                (Display contents of string.
)                                       The string is empty.)
```

For another example, see the SPACEX.CLI macro near the end of Chapter 4.

Command Summary Information

In the CLI commands, macros, and pseudomacros that follow, summary information appears in a list at the end of the format descriptions. The summary information answers the following questions:

- Can you use template characters?
- Does the command, pseudomacro, or macro accept argument switches?
- What privilege is necessary to use the command, pseudomacro, or macro? The following terms are used to answer this question:

Standard	Available to any user, unless specifically restricted.
PID 2	Only PID 2, the master CLI at the system console, can perform the function.
System Operator	Only a process with the username OP can perform the function. This applies to PID 2 and any son process.
System Manager	Your user profile must grant System Manager privilege and you must turn it on. This privilege also lets you perform certain operations otherwise restricted to PID 2.
Superuser	You must have Superuser mode on (which is possible only if your user profile grants the Superuser privilege).
Superprocess	You must have Superprocess mode on (which is possible only if your user profile grants the Superprocess privilege.)

- What other commands, pseudomacros, macros, or utilities perform related functions?

Utility programs do not have these topics, since access to them is controlled by the program file access control list (the ACL to file program−pathname.PR).

Getting Help

If in doubt about a CLI command, pseudomacro, or macro, you can ask the CLI to display an explanation of the item.

To display very brief help for a command, type HELP followed by the item's name. For example

```
) HELP DELETE)
```

The resulting display reports whether the DELETE command accepts or requires arguments, and lists the switches you can use with the command.

Often, though, you'll need more verbose help. In that case you can either add the /V switch to the HELP command (HELP/V TIME or HELP/V !TIME) or use the HELPV macro (HELPV DELETE). The HELPV macro works the same way as the HELP/V command.

For help with macros or utility programs, type HELP followed by the item's name prefixed with an asterisk (HELP/V *BROWSE).

Summary of CLI Commands, Pseudomacros, Macros, and Utility Programs

Table 5–1 summarizes differences between the CLI32 and CLI16 versions of each CLI command, pseudomacro, macro. Utility programs behave exactly the same way under both CLIs, but are also summarized in this alphabetical listing.

Table 5–1 Summary of Commands, Macros, Pseudomacros, and Utilities

Command, Macro, Pseudomacro, Utility	What it does	Differences between CLI16 and CLI32
ACL command	Displays or sets the access control list for a file.	CLI32 has switch enhancements, filtering and template control.
!ACL pseudomacro	Expands the access control list for the specified file.	No difference.
!ARGUMENT pseudomacro	Expands to the requested arguments.	Only CLI32 has this pseudomacro.
!ASCII pseudomacro	Expands to the ASCII character from an octal value	No difference.
ASSIGN command	Assigns a character device for your exclusive use.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
BIAS command	Displays or sets the system's bias factor.	CLI32 has switch enhancements.
BLOCK command	Suspends a process.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
BRAN utility	Analyzes a program break file.	No difference.
BREAKFILE command	Displays or sets the break file format for a process.	CLI32 has switch enhancements.
BROADCAST.CLI macro	Sends a message to all users on your system.	No difference.
BROWSE utility	Types, displays, or finds strings in an ASCII or binary file.	No difference.
BYE command	Terminates this CLI process.	CLI32 has switch enhancements.
CHAIN command	Replaces the current CLI process with a specified program.	CLI32 has switch enhancements.
CHARACTERISTICS command	Displays or sets the characteristics for a character device.	CLI32 has switch enhancements.

Page 1 of 13

Table 5–1 Summary of Commands, Macros, Pseudomacros, and Utilities

Command, Macro, Pseudomacro, Utility	What it does	Differences between CLI16 and CLI32
CHECKTERMS command	Checks for a termination message from a son process.	CLI32 has the /STR= and /ESTR= switches; CLI32 accepts PID as argument; otherwise, no difference.
CLASS1 command	Displays or sets the Class 1 severity level.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
CLASS2 command	Displays or sets the Class 2 severity level.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
CLEARDEVICE command	Simulates a CTRL–Q (X–ON) from a device, or sends a break character to a device.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
!CLI pseudomacro	Expands to CLI32 or CLI16, depending on the CLI running.	For CLI32, returns CLI32; for CLI16, returns CLI16.
CLOSE command	Closes a file.	Only CLI32 has this command.
COMMENT command	Starts a comment line in a macro.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
CONNECT command	Creates a connection with a server process.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
CONINFO command	Displays console and session addressing information.	Only CLI32 with AOS/VS II has this command.
!CONSOLE pseudomacro	Expands to console filename or batch queue name.	No difference.
CONTROL command	Sends a control message to a process.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
CONVERT utility	Converts an RDOS .RB file to an AOS/VS .OB file.	No difference.
COPY command	Copies one or more files to a destination file.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
CPIO_VS utility	Dumps, loads, or copies files in UNIX cpio format.	No difference.

Page 2 of 13

Table 5–1 Summary of Commands, Macros, Pseudomacros, and Utilities

Command, Macro, Pseudomacro, Utility	What it does	Differences between CLI16 and CLI32
CPUID command	Displays the central processing unit identifier.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
CREATE command	Create a file.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
CURRENT command	Displays settings in the current CLI environment.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
DATAFILE command	Displays or sets the data file pathname.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
!DATAFILE pseudomacro	Expands to the pathname of the current data file.	CLI32 has the /LEVEL= and /PREVIOUS= switches.
DATE command	Displays or sets the system date.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
!DATE pseudomacro	Expands to the current date.	CLI32 has the /NUMERIC switch.
DEASSIGN command	Releases a previously assigned character device.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
DEBUG command	Runs a program in the assembly language debugger.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
!DECIMAL pseudomacro	Expands to the decimal equivalent of an octal number.	No difference.
DEFACL command	Displays or sets the default access control list.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
!DEFACL pseudomacro	Expands to the current default access control list.	CLI32 has the /LEVEL= and /PREVIOUS= switches.
DELETE command	Deletes one or more files.	CLI32 has enhancements to switches and templates.
DIRECTORY command	Displays or sets the working directory.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.

Page 3 of 13

Table 5–1 Summary of Commands, Macros, Pseudomacros, and Utilities

Command, Macro, Pseudomacro, Utility	What it does	Differences between CLI16 and CLI32
!DIRECTORY pseudomacro	Expands to the full pathname of the working directory.	CLI32 has the /LEVEL= and /PREVIOUS= switches.
DISCONNECT command	Breaks connection with a server process.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
DISMOUNT command	Asks the system operator to remove a tape from a unit.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
DISPLAY utility	Displays the contents of binary and tape files; converts EBCDIC to ASCII.	No difference.
DUMP command	Dumps files from the working directory to tape or disk.	Only CLI16 has this command. (There is a macro to run DUMP_II.)
DUMP_II utility	Dumps files from the working directory to tape or disk.	No difference.
!EDIRECTORY pseudomacro	Expands to the directory portion of a pathname.	No difference.
!EEXTENSION pseudomacro	Expands to the filename suffix portion of a pathname.	No difference.
!EFILENAME pseudomacro	Expands to the filename portion of a pathname.	No difference.
!ELSE pseudomacro	Begins a command sequence that executes when a condition is false.	No difference.
!ENAME pseudomacro	Expands to the name portion of a pathname.	No difference.
!END pseudomacro	Ends a command sequence that begins with a conditional pseudomacro.	No difference.
!EPREFIX pseudomacro	Expands to the prefix portion of a pathname.	No difference.
!EQUAL pseudomacro	Compares two values and continues execution based on the result.	No difference.
EXECUTE command	Executes a program.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.

Page 4 of 13

Table 5–1 Summary of Commands, Macros, Pseudomacros, and Utilities

Command, Macro, Pseudomacro, Utility	What it does	Differences between CLI16 and CLI32
!EXIT pseudomacro	Terminates current macro, current and calling macros, or macro loop, depending on switches specified.	CLI32 only.
!EXPLODE pseudomacro	Expands arguments into single characters.	No difference.
FCU utility	Sets nonstandard form parameters for files to be printed.	No difference.
FILCOM utility	Compares two binary files.	No difference.
!FILENAMES pseudomacro	Expands to one or more filenames.	CLI32 has enhancements to switches and templates.
FILESTATUS command	Displays file information.	CLI32 has enhancements to switches and templates.
GROUPLIST command	Displays or sets your group membership list.	Only CLI32 with AOS/VS II has this command.
!GROUPLIST pseudomacro	Expands to your current group list.	Only CLI32 with AOS/VS II has this pseudomacro.
HELP command	Displays information about a CLI command, pseudomacro, or topic.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
HELPV macro	Displays verbose information about a CLI command, pseudomacro, or topic	No difference.
!HID pseudomacro	Expands to a network host ID.	No difference.
HISTORY command	Displays and retrieves previous command lines.	Only CLI32 has this pseudomacro.
HOST command	Displays a system host name.	CLI32 has the /STR=, /ESTR= and /STRING switches.
!HOST pseudomacro	Expands to a host name.	No difference.
!IMPLODE pseudomacro	Removes space or tab separators between arguments.	Only CLI32 has this pseudomacro.
!INDEX pseudomacro	Expands to the location of one string in another string.	Only CLI32 has this pseudomacro.
INITIALIZE command	Adds a logical disk unit (LDU) to the working directory.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.

Page 5 of 13

Table 5–1 Summary of Commands, Macros, Pseudomacros, and Utilities

Command, Macro, Pseudomacro, Utility	What it does	Differences between CLI16 and CLI32
JPINITIALIZE command	Initializes a job processor.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
JPRELEASE command	Releases an initialized job processor.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
LABEL utility	Write a label to a magnetic tape or diskette.	No difference.
LDUINFO utility	Displays status information on a disk unit or LDU.	Supplied with AOS/VS II only.
!LENGTH pseudomacro	Expands to the lengths (in characters) of arguments.	Only CLI32 has this pseudomacro.
LEVEL command	Displays the number of the current environment level.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
!LEVEL pseudomacro	Expands to the number of the current environment level.	CLI32 has the /LEVEL= and /PREVIOUS= switches.
LISTFILE command	Displays or sets the current list file pathname.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
!LISTFILE pseudomacro	Expands to the full pathname of the current list file.	CLI32 has the /LEVEL= and /PREVIOUS= switches.
LOAD command	Loads files from the tape or disk into the working directory.	Only CLI16 has this command. (There is a macro to run LOAD_II.)
LOAD_II utility	Loads files from the tape or disk into the working directory.	No difference.
LOCALITY command	Assigns a new user locality to a process.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
LOCK command	Locks the CLI.	With CLI16, this is available in LOCK_CLI only; with CLI32, it is always available – with major differences.
LOGEVENT command	Enters a message in the system log file.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.

Page 6 of 13

Table 5–1 Summary of Commands, Macros, Pseudomacros, and Utilities

Command, Macro, Pseudomacro, Utility	What it does	Differences between CLI16 and CLI32
LOGFILE command	Displays or sets the user log file.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
LOGOFFMACRO command	Sets or displays the log-off macro filename.	Only CLI32 has this command.
!LOGON pseudomacro	Expands to CONSOLE, BATCH, or null, depending on how you logged on.	No difference.
!LOOPEND pseudomacro	Ends the sequence of CLI commands introduced with !LOOPSTART.	CLI32 only.
!LOOPSTART pseudomacro	Begins an iterative sequence of CLI commands.	CLI32 only.
MESSAGE command	Displays the text message that corresponds to an error code.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
MIRROR command	Starts synchronizing an image of a mirrored LDU.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
MOUNT command	Asks the system operator to mount a tape on a tape unit.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
MOVE command	Moves a copy of one or more files to a different directory.	CLI32 has enhancements to switches and templates.
!NEQUAL pseudomacro	Compares two values and continues execution based on the result.	No difference.
!OCTAL pseudomacro	Expands to the octal equivalent of a decimal number.	No difference.
OPEN command	Opens a file for input or output.	Only CLI32 has this command.
OPERATOR command	Displays or sets the status of the CLI's ability to use labeled diskettes.	Only CLI16 has this command.
!OPERATOR pseudomacro	Expands to ON or OFF, depending on whether a system operator is on duty.	No difference.
PASSWORD command	Sets a password for your CLI process.	Only CLI32 has this command.

Page 7 of 13

Table 5–1 Summary of Commands, Macros, Pseudomacros, and Utilities

Command, Macro, Pseudomacro, Utility	What it does	Differences between CLI16 and CLI32
PATHNAME command	Displays the full pathname of a file.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
!PATHNAME pseudomacro	Expands to the full pathname of a file.	CLI32 accepts multiple arguments.
PAUSE command	Delays the CLI by the given number of seconds.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
PERFORMANCE command	Displays information about this CLI process.	Only CLI16 has this command.
PERMANENCE command	Displays or sets a file's permanence setting.	CLI32 has enhancements to switches and templates.
!PID pseudomacro	Expands to the process ID of this CLI.	No difference.
!PIDS pseudomacro	Expands to all process IDs on your system.	No difference.
POP command	Returns to the previous CLI environment level.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
PREFIX command	Displays or sets the CLI prefix string.	CLI32 has /HISTORY=, /7BIT, and other switch enhancements.
PREVIOUS command	Displays the settings for the previous CLI environment level.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
PRIORITY command	Displays or sets the priority of a process.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
PRIVILEGE command	Sets or displays Superuser, Superprocess, and System Manager privilege settings.	No difference.
PROCESS command	Creates a process.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
PROMPT command	Displays or sets the current CLI prompt commands.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.

Page 8 of 13

Table 5–1 Summary of Commands, Macros, Pseudomacros, and Utilities

Command, Macro, Pseudomacro, Utility	What it does	Differences between CLI16 and CLI32
PRTYPE command	Displays or sets the type of a subordinate process.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
PUSH command	Moves down to the next CLI environment level.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
QBATCH command	Creates and submits a job to a batch queue.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
QCANCEL command	Cancels a waiting or active job in a batch or print queue.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
QDISPLAY command	Displays queue information.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
QFTA command	Submits an entry to the File Transfer Agent queue.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
QHOLD command	Holds an entry in its queue.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
QMODIFY command	Changes information about an entry in a queue.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
QPLOT command	Submits a job to a print queue.	CLI32 has enhancements to switches and templates.
QPRINT command	Submits a job to a print queue.	CLI32 has enhancements to switches and templates.
QSNA command	Submits a job to the Systems Network Architecture queue.	CLI32 has enhancements to switches and templates.
QSUBMIT command	Submits a job to a batch or spool queue.	CLI32 has enhancements to switches and templates.
QUNHOLD command	Frees an entry currently held in a queue.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
RDOS utility	Loads, dumps, or converts files in RDOS format.	No difference.
READ command	Reads and displays a line from a file.	Only CLI32 has this command.

Page 9 of 13

Table 5–1 Summary of Commands, Macros, Pseudomacros, and Utilities

Command, Macro, Pseudomacro, Utility	What it does	Differences between CLI16 and CLI32
!READ pseudomacro	Displays a text string and expands to the typed response.	No difference.
RELEASE command	Removes a logical disk unit (LDU) from the working directory.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
RENAME command	Changes the name of a file.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
REPORT utility	Displays log file records.	No difference.
REVISION command	Displays or sets a program revision number.	CLI32 has enhancements to switches and templates.
REWIND command	Rewinds one or more magnetic tapes.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
RUNTIME command	Displays runtime information for a process.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
SCOM utility	Compares two ASCII text files and displays differences.	No difference.
SCREENEDIT command	Displays or sets your Screenedit mode.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
SEARCHLIST command	Displays or sets your search list.	CLI32 has switch enhancements.
!SEARCHLIST pseudomacro	Expands to the current searchlist.	CLI32 has the /LEVEL= and /PREVIOUS= switches.
SEND command	Sends a message to a terminal.	CLI32 has the /7BIT, /STR=, and /ESTR= switches; otherwise, no difference.
!SIZE pseudomacro	Expands to the length of a file in bytes.	No difference.
!SONS pseudomacro	Expands to a list of subordinate process IDs.	No difference.
SPACE command	Displays or sets the disk space allotment for a control point directory or LDU.	CLI32 has enhancements to switches and templates. With AOS/VS II, it can display usage figures for standard directories.

Page 10 of 13

Table 5–1 Summary of Commands, Macros, Pseudomacros, and Utilities

Command, Macro, Pseudomacro, Utility	What it does	Differences between CLI16 and CLI32
SQUEEZE command	Displays or sets the Squeeze mode.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
STRING command	Displays or sets the value of a string.	CLI16 has only one string per level. CLI32 has multiple, named strings, accessible by switch.
!STRING pseudomacro	Expands to the value of a string.	CLI16 has only one string per level. CLI32 has multiple, named strings, accessible by switch.
!SUBSTRING pseudomacro	Expands to the specified part of a text string.	Only CLI32 has this pseudomacro.
SUPERPROCESS command	Displays or sets the Superprocess mode.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
SUPERUSER command	Displays or sets the Superuser mode.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
SYSID command	Displays or sets your system identifier.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
SYSINFO command	Displays information about the current system.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
SYSLOG command	Displays or sets the status of system logging.	CLI32 required for /VERBOSE, CON0 logging, and Superuser logging. CLI32 has the /STR= and /ESTR= switches.
!SYSTEM pseudomacro	Expands to the name of the operating system.	No difference.
TAR_VS utility	Dumps or loads files in UNIX tar format.	No difference.
TERMINATE command	Terminate a process.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.

Page 11 of 13

Table 5–1 Summary of Commands, Macros, Pseudomacros, and Utilities

Command, Macro, Pseudomacro, Utility	What it does	Differences between CLI16 and CLI32
TIME command	Displays or sets the system time.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
!TIME pseudomacro	Expands to the system time.	CLI32 has the /NUMERIC switch.
TRACE command	Displays or sets trace mode for debugging CLI macros.	CLI32 has major enhancements.
TREE command	Lists the father and son processes of a process.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
TYPE command	Displays the contents of a file.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
!UADD pseudomacro	Expands to the sum of two integers.	CLI32 accepts multiple arguments.
!UDIVIDE pseudomacro	Expands to the quotient of an argument divided by another.	CLI32 accepts multiple arguments.
!UEQ pseudomacro	Tests two unsigned integer arguments for equality.	No difference.
!UGE pseudomacro	Tests the first argument for greater or equal value to the second.	No difference.
!UGT pseudomacro	Tests the first argument for greater value than the second.	No difference.
!ULE pseudomacro	Tests the first argument for lesser or equal value to the second.	No difference.
!ULT pseudomacro	Tests the first argument for lesser value than the second.	No difference.
!UMAXIMUM pseudomacro	Expands to the maximum value of the arguments.	Only CLI32 has this pseudomacro.
!UMINIMUM pseudomacro	Expands to the minimum value of the arguments.	Only CLI32 has this pseudomacro.
!UMODULO pseudomacro	Expands to the value of the first argument modulo the second argument.	CLI32 accepts multiple arguments.
!UMULTIPLY pseudomacro	Expands to the product of two numbers.	CLI32 accepts multiple arguments.

Page 12 of 13

Table 5–1 Summary of Commands, Macros, Pseudomacros, and Utilities

Command, Macro, Pseudomacro, Utility	What it does	Differences between CLI16 and CLI32
UNBLOCK command	Unblocks a previously blocked process.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
!UNE pseudomacro	Tests two integer arguments for inequality.	No difference.
UNLOCK command	Frees a locked CLI.	With CLI16, this is available in LOCK_CLI only; with CLI32, it is always available – with major differences.
!USERNAME pseudomacro	Expands to the username of the CLI.	No difference.
!SUBTRACT pseudomacro	Expands to the difference between two integer values.	CLI32 accepts multiple arguments.
VAR command	Displays or sets the value of CLI variable VAR/NAME=.	Only CLI32 has this command.
!VAR pseudomacro	Expands to the current value of VAR/NAME=.	Only CLI32 has this pseudomacro.
VARn command	Displays or sets the value of CLI variable VARn.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
!VARn pseudomacro	Expands to the current value of VARn.	CLI32 has the /LEVEL= and /PREVIOUS= switches.
WHO command	Displays information on one process	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.
WHOS.CLI macro	Displays information on all processes on the system.	No difference.
WRITE command	Displays arguments or writes them to a file.	CLI32 has the /7BIT, /FILE, /FORCE, /NONEWLINE, /STR=, and /ESTR= switches.
XEQ command	Executes a program.	CLI32 has the /STR= and /ESTR= switches; otherwise, no difference.

Page 13 of 13

ACL

Command

Displays or sets the access control list for a file.

Format

ACL pathname $\left[\begin{array}{l} \textit{username, access-types [...]} \\ \textit{username[:groupname][,...],access-types [...]}^{1} \end{array}\right]$

^{1} CLI32 with AOS/VS II only.

If you specify only pathname, this command displays the access control list for the specified files; you can use a template for the pathname.

If you specify an ACL, the command assigns that ACL to the file. An ACL consists of a username (which can be a template) and one or more access types (or no type if you want to deny all access for that username). You can separate the username from the access types by a comma or space; for example ACL MYFILE JONB,OWARE +,RE. Access types are explained in Chapter 2. A summary follows.

Access Type	Represents
O	Owner access
W	Write access
A	Append access
R	Read access
E	Execute access

You can supply more than one username/access type pair. If you do, and you use username templates, place the more specific usernames before the more general ones. This is needed because the system assigns access to the files in the order of the usernames you have supplied. So if a username occurs more than once, the first entry that matches the username will apply. For example, assume user SMITH sets the ACL of his file FILEX as follows:

```
ROMERO,WR   SM+,R   MCDONALD,R   SMITH,OWARE
```

The system imposes SM+,R because it is first, and ignores SMITH,OWARE. SMITH will have only Read access to his own file. If he inserts the specific usernames before the general ones, he will retain the OWARE access he wants. The correct order is

```
SMITH,OWARE  ROMERO,WR  MCDONALD,R  SM+,R
```

If you specify an ACL using the # template, the system tries to assign it from the bottom level up; for example, with pathname MYDIR:MYFILE, it will try to assign the ACL to MYFILE, and then MYDIR. This can cause an ACL command with # to fail if you lack Write access to lower level directories (even though you have Write access to higher level directories). With CLI32, you can avoid this problem by using the /TOPDOWN switch. You can overcome any ACL with Superuser on.

With AOS/VS II and CLI32, a username can include a group name of the form username:groupname,access. For group access to work, there must be a file named groupname in directory :GROUPS, and the username must be defined in that file. User groups are further explained in Chapter 2 and in *Managing AOS/VS and AOS/VS II.*

CLOSE

Command

Closes a file (CLI32 only).

Format

```
CLOSE  { /FILEID=file-ID }
       { /ALL            }
```

With the /ALL switch, this command closes all files that you previously opened with the OPEN command.

With the /FILEID= switch, closes the file that the switch identifies. When you open a file with the OPEN command, the CLI displays each file's identifier; you must give the identifier to the CLOSE command to close the file.

- No templates.
- Requirement: *Standard.*
- See also: OPEN, READ, !READ, WRITE.

Why Use It?

Use the CLOSE command to close a file after you have opened, and most likely read from or written to, it.

Command Switches

The section "Universal CLI Switches," earlier in this chapter, describes switches /1, /2, /L, /L=pathname, and /Q, which you can use with all commands. That section also explains the CLI32 switches /STR= and /ESTR=.

/ALL	Closes all files that OPEN commands previously opened. If you supply this switch, you cannot give any arguments to the CLOSE command.
/FILEID=file-ID	Identifies the file you want to close. The default file ID is the file's name (not pathname), without any trailing suffix. You can learn the file IDs of all open files by typing the OPEN command without an argument.

CLOSE Example

```
) OPEN/READ FILE_ABC)
FILE_ABC
) READ/FILEID=FILE_ABC)
This is the first line of a file named FILE_ABC.
) CLOSE/FILEID=FILE_ABC)
```

Also, see the three examples in the explanation of the OPEN command.

COMMENT

Command

Includes a comment in a CLI macro file.

Format

COMMENT *[text]*

This command lets you insert text into a CLI macro file. The CLI generally ignores this text when it executes the macro. You can also give a comment as a command line.

- No templates.
- No argument switches.
- Requirement: *Standard.*

Why Use It?

Use the COMMENT command to provide commentary within a CLI macro file. Comments can help explain the purpose of a command sequence or the meaning of arguments. A brief commentary will also make it easier for someone else to maintain or update the macro file.

However, the CLI lets you write comments in other ways; for example, via \\ in CLI32. For more information, see the section "Using Comments in Macros" in Chapter 1.

Restrictions

The CLI interprets the text argument for this command as it would any other command. If the text contains parentheses, angle brackets, or square brackets, the CLI will try to expand their contents as usual.

Generally, avoid using parentheses, angle brackets, square brackets, and semicolons in comment lines. A semicolon terminates the COMMENT command; the CLI will treat any text that follows the semicolon as a new command.

With CLI32, you can give *lexical comments*. A lexical comment is a string of characters that begins immediately after the two-character string \\. The lexical comment ends with a delimiter. Lexical comments accept any characters, specifically including angle brackets, square brackets, parentheses, colons, and semicolons. The only character that the CLI interprets after the lexical indicator is an ampersand (&). For clarity, you can precede the \\ characters with the word COMMENT. For example

```
COMMENT \\ Change directory; then delete file.
DIRECTORY %1%
DELETE %2%
```

Or, with either CLI, you can use a conditional operator comment. A conditional operator comment consists of a conditional operator that never returns True, followed by text and an [!END] terminator. Example 3 shows this method of including extensive comments with no restriction on characters used and without having to type COMMENT or \\ at the start of each line.

COMMENT Command Switches

The section "Universal CLI Switches," earlier in this chapter, describes switches /1, /2, /L, /L=pathname, and /Q, which you can use with all commands. That section also explains the CLI32 switches /STR= and /ESTR=.

COMMENT Example 1

```
) create/i  my_macro.cli↵
)) comment Test the first argument and current string for equality↵
)) [!equal, %1%, [!string] ]↵
))           comment The values are equal — perform the following steps.↵
))           ...↵
)) [!else]↵
))           comment The values are not equal — perform these steps.↵
))           ...↵
)) [!end]↵
)) )↵
```

In this example, we create a macro in file MY_MACRO.CLI and input the lines directly. The macro uses the COMMENT command to explain its use of the !EQUAL and !ELSE pseudomacros. When we execute this macro, the CLI ignores the COMMENT commands.

COMMENT Example 2

```
COMMENT This is macro FAS.CLI.
PUSH; PROMPT POP
STRING A\\B
[!EQUAL,[!STRING],A]
            COMMENT CLI32 is executing and /COUNT is valid.
            FILESTATUS/ASSORTMENT/SORT/COUNT%/%  %-%
[!ELSE]
            COMMENT CLI16 is executing and /COUNT is invalid.
            FILESTATUS/ASSORTMENT/SORT%/%  %-%
[!END]
POP
```

The third line of macro FAS.CLI uses a lexical comment to determine which CLI is executing. If it is CLI32, the CLI String contains the single character A. If it is CLI16, the String contains the four characters A\\B because CLI16 does nothing special with \\.

COMMENT Example 3

```
comment  Macro CSEA.CLI to change search list.
comment
[!equal,large,comment]

This macro will either add one or more directories at the beginning of
your search list or remove one directory from your search list.

Invoke the macro with the /ADD switch to add the directories given
given by %1-% to your search list.  For example,

) CSEA/ADD :UTIL:PRESENT :UTIL:F77

Invoke the macro without a switch to remove the directory given
by %1% from your search list.  For example,
) CSEA :UTIL:BASIC
...
[!end]
.
```

This is the beginning of a macro. Notice how you can place multiple–line comments between an untrue !EQUAL conditional an !END. Macros with this !equal ... !end construction can also contain bracket characters — [(< — and semicolons (;) as part of the comment text.

CONINFO *Command*

Displays console and session addressing information (CLI32 with AOS/VS II only).

Format

CONINFO *[console ...]*

CONINFO displays the addressing information for the specified console or, if you do not supply any arguments, for the current console.

- No templates.
- No argument switches.
- Requirement: *Standard* to view information for your own console;
  PID2 or *System Manager* to view information for other consoles.

The following table lists the information displayed for those connection types that the command supports. Using CONINFO for an unsupported type causes an ERIFD error, Invalid Function for this Device.

Supported Connection Types	Information Displayed
ITC/LTC over TCP/IP	Device code, engine and line numbers, IP address, and port number.
ITC/LTC over XNS	Device code, engine and line numbers, and CS200 Ethernet address.
ITC/PVC	Device code, engine and line numbers, and either an address or, if the PVC subtype specifies NAME, an ASCII string.
R0[1] Telnet	Line number, IP address, and port.
IACs	Device code, engine and line numbers, and modem flag if applicable.
R0 DUARTs	Device code, engine and line numbers, and CON0 flag if applicable.
Opcon – TTI/TTO	Device code, engine and line numbers, and CON0 flag.

[1]Ring zero.

Types ITC and LTC return network address only when a connection exists. Otherwise, they return device code, engine number, and line number. An R0 Telnet console returns only line number when no connection exists.

Why Use It?

Use the CONINFO command to get addressing information about a console.

CONINFO Example 1

The following example uses the CONINFO command for the current console.

```
) WHO↵
PID:     55    WARREN     CON156        :CLI32.PR
) CONINFO↵
@CON156        Device code: 71      Engine: 2     Line: 4
```

CONINFO Example 2

This example shows an attempt to use the CONINFO command without privilege to obtain information about another console.

```
) CONINFO @CON157↵
Warning: Caller not privileged for this action
```

CONINFO Example 3

This final example turns on the System Manager privilege prior to using CONINFO to request information about another console.

```
) PRIVILEGE SYSTEMMANAGER ON↵
Sm) CONINFO @CON157↵
@CON157        Device code: 71      Engine: 1     Line: 2
```

COPY

Command

Copies one or more files to a destination file.

Format

COPY destination–file sourcefile *[...]*

This command copies the source file(s) – in the order specified – to the destination file. The destination file may be a disk file, peripheral device file (such as a magnetic tape file), or printer queue (such as @LPT). It cannot be a directory file.

Depending on your use of command switches, you can create the destination file, append to an existing file, or replace an existing file. If the second or subsequent source file does not exist, the CLI issues a warning message and continues copying until it has copied all the files you specified.

- No templates.
- No argument switches.
- Requirement: *Standard.*
- See also: MOVE, DUMP, LOAD/LOAD_II and, for AOS/VS II, RENAME.

Why Use It?

Use the COPY command to add text to a file or build one large file from smaller ones. With it, you can duplicate the contents of a diskette or tape file onto a disk file, or vice versa. The command will also duplicate a file within a directory. Because it copies the *contents* of a file, without the name, creation date, and so on, you might want to use it instead of the MOVE command.

The COPY command does not copy the ACL(s), time–date created, or User Data Area (UDA) of the source file(s); if the CLI creates a new file, the file has the current date–time created and the default ACL. If you want to retain the existing ACL and date–time created, use MOVE, not COPY.

In AOS/VS II, if you want to relocate the original file to a different directory on the same LDU (instead of creating a copy there), you can use the RENAME command to change the pathname of the file.

COPY Command Switches

The section "Universal CLI Switches," earlier in this chapter, describes switches /1, /2, /L, /L=pathname, and /Q, which you can use with all commands. That section also explains the CLI32 switches /STR= and /ESTR=. Note that /STR= does not work as usual — it returns null — with the COPY and TYPE commands.

/A	Appends the source file(s) to an existing destination file.
/APPEND	(CLI32 only.) Same as /A.

/B	Sets binary mode (for character devices) so that special characters are not interpreted or translated.
/BINARY	(CLI32 only.) Same as /B.
/BACKUP	Uses checkpointing in a XODIAC File Transfer Agent (FTA) transfer. If the system aborts the file transfer, it returns a unique identification (UID) for use with the /BACKUP=uid switch. (Meaningful only with the /FTA switch.)
/BACKUP=uid	Enables you to recover from an aborted FTA file transfer if you have used the /BACKUP switch. You must supply the unique identifier (UID) that /BACKUP provides. (Meaningful only with the /FTA switch.)
/COMPRESS	Uses compression when copying files with FTA. (You must use /FTA with this switch.)
/D	Deletes the specified destination file (which must exist) and creates a new file with the same name and specifications.
/DELETE	(CLI32 only.) Same as /D.
/FTA	Uses XODIAC FTA to copy the file(s). The destination system uses each source file's creation date and time. Only the /A, /BACKUP[=uid], /COMPRESS, /D, and /V switches are compatible with /FTA.
/IDENSITY=density	Reads from magnetic tape at specified density, overriding system default density. Options for density are 800, 1600, 6250 (bytes per inch), or ADM (Automatic Density Matching); LOW, MEDIUM (reverts to LOW on a dual-density unit), or HIGH.
/IMTRSIZE=bytes	Specifies a nondefault magnetic tape buffer size for reading (input). To read a file created on an RDOS system with the XFER command, use 510.
/ODENSITY=density	Writes at specified density to magnetic tape. Options for density are 800, 1600, 6250 (bytes per inch), or ADM (Automatic Density Matching); LOW, MEDIUM (reverts to LOW on a dual-density unit), or HIGH. If you intend to use a tape on another unit, be careful about choosing LOW, MEDIUM, or HIGH: "low" or "high" on one unit may not be compatible with the values on another unit, which would prevent reading from the tape. The default is the value selected for the tape unit during VSGEN. If you are copying to labeled tape or to a tape file other than file 0, the system enforces the density already used on the tape. (If you specify a density that conflicts with the current density, the command will fail.)

/OMTRSIZE=bytes	Specifies a magnetic tape buffer size for writing (output). Overrides the default. To read this tape, you must use the buffersize specified with the /IMTRSIZE switch. To create a file that can be read (by the XFER command) on an RDOS system, specify 510.
/V	Displays the pathname of each source file copied.
/VERIFY	(CLI32 only.) Same as /V.

COPY Example 1

```
) DELETE/2=IGNORE MYFILE.BACKUP)
) COPY MYFILE.BACKUP MYFILE)
```

These commands create a new file called MYFILE.BACKUP and copy the contents of MYFILE into it.

COPY Example 2

```
) COPY MY_NOVEL CHAPTER1)
) COPY MY_NOVEL CHAPTER2 CHAPTER3)
Warning: File already exists, File MY_NOVEL
) COPY/A MY_NOVEL CHAPTER2 CHAPTER3)
```

The first command creates a new file called MY_NOVEL and copies the content of CHAPTER1 into it. The next command attempts to copy additional files to CHAPTER1, but the CLI rejects the command. Finally, COPY with the /A switch appends the contents of CHAPTER2 and CHAPTER3 to the file.

COPY Example 3

```
) COPY/V MY_DATA @MTB0:0)
@MTB0:0
```

This command copies the contents of magnetic tape file 0 to the working directory and verifies the sourcefile name. This works properly if the file was originally copied to tape with the COPY (not DUMP/DUMP_II) command.

COPY Example 4

```
) COPY/V/FTA MAY.REPORT :NET:SYS3:UDD:SANDY:REPORTS:MAY.REPORT)
:NET:SYS3:UDD:SANDY:REPORTS:MAY.REPORT
```

This example copies file MAY.REPORT from a different host system to the working directory. It uses the network agent FTA. The user needs the same username/password pair on both of the networked systems.

COPY Example 5

```
) COPY/V/FTA/RECENT  :NET:TITAN:UDD:TERRY:REPORT  REPORT⤶
:NET:TITAN:UDD:TERRY:REPORT
```

This command copies the file REPORT (in the working directory) to a file called REPORT on a remote host named TITAN. The /DELETE switch causes the system to replace the target file if it already exists. As above, the user must have the same username and password on the local and remote systems (as well as appropriate access to the target file and its directories).

CPIO_VS *Utility*

Dumps files from the working directory in UNIX cpio format, loads from a cpio–format dump file, or copies files.

Format

To dump files:

```
        CPIO_VS/OUTPUT/DEVICE=dumpfilename/DATA=file–with–files
```

To load files:

```
        CPIO_VS/INPUT/DEVICE=dumpfilename  [pathname] [...]
```

To copy files:

```
        CPIO_VS/PASS    directory–pathname
```

The CPIO_VS utility produces and reads dump files in the cpio Archive/Interchange File Format specified in IEEE Std. 1003.1–1988. Primarily, CPIO_VS is intended to let you transfer files between an AOS/VS or AOS/VS II system and a UNIX® system — perhaps a DG/UX™ system running on an AViiON® workstation.

- *To create dump (archive) files* to be read on UNIX systems, use the first format. The dump filename can be a tape unit devicename (for example, @MTB0:0) or, if you want to dump to disk, a disk filename. You can specify the files by building their names into the file-with-files; for example, by typing the command

  ```
  WRITE/L=DFILE [!FILENAMES MYFILE+]
  ```

  followed by a CPIO_VS command of the form

  ```
  CPIO_VS/OUTPUT..../DATA=DFILE
  ```

  If you omit an argument, the utility will wait for you to specify the filenames you want dumped. Type each filename, followed by NEW LINE. When you have specified all the files you want, terminate the list by pressing CTRL–D.

- *To load a dump (archive) file,* use the second format. CPIO_VS can read an archive file created by the cpio utility on a UNIX system or by CPIO_VS. To be readable from a multicapacity cartridge tape drive (models 6675, 6676, 6677, or 7656), the file must have been created with the blocking switch (–B for the UNIX cpio command or /BLOCK for CPIO_VS) specified. The dump filename can be a tape unit devicename (for example, @MTB0:0) or a disk filename. The program accepts template characters on a load.

- *To copy files* from the working directory to another directory, use the third format. You supply names interactively. Type each name, followed by NEW LINE; terminate the list with CTRL-D.

CPIO_VS (continued)

While loading, if a file in the dump file has the same name as a file in the working directory, the CPIO_VS utility compares the date/time last modified of the two files. If the file in the working directory is newer (has a later date/time last modified), the utility displays a message of the form *xxx, Newer file exists* (xxx is the filename) and does not load the file. If the file in the dump file has a later date/time last modified, or the same date/time last modified, then the utility deletes the file in the working directory and loads the file in the dump file. If you want to load files unconditionally, regardless of their date/time modified, use the /UNCONDITIONAL switch.

The CPIO_VS utility converts characters and access permissions as detailed later. It reports disk blocks in 512-byte quantities (as usual for AOS/VS and AOS/VS II); pathnames are limited to 256 characters.

On most errors, CPIO_VS will report the cause and continue to copy other files. It will skip any unrecognized files it encounters in the dump file. If you try to load from a dump file not in cpio format, the utility will display the message *Unrecognizable archive* and stop.

You can abbreviate a switch name to the smallest number of characters needed to identify it. Usually this is one character. Characters in switches are not case sensitive.

The file CPIO_VS.PR is actually a link to file TAR_VS.PR, but the program functions just as if it were a separate CPIO program.

- Accepts templates on loads (CPIO_VS/INPUT) only.
- Accepts utility switches (described later).
- Requirement: *Standard*. You need Execute access to the program file TAR_VS.PR in :UTIL, since CPIO_VS.PR is a link to this file. To dump files, you need Execute access to any directory from which you want to dump and Read access to any file you want to dump. To load or copy into a directory, you need Write or Append and Execute (WE or AE) access to the directory.
- See also: TAR_VS.

Why Use It?

Use CPIO_VS to create cpio-format dump files to be read by the cpio utility on a UNIX system, or use it to load files dumped by cpio on a UNIX system. If you are familiar with UNIX, you may want to use the CPIO utility instead of CPIO_VS. (If you use CPIO, you can use familiar UNIX syntax but must use AOS/VS or AOS/VS II device names; also, be aware that the CPIO utility provides only those features that CPIO_VS does.)

Generally, use CPIO_VS for individual files (although it does work for multiple files). To dump or load large numbers of files, use the TAR_VS utility.

Generally, to create dump files to be read on an AOS, AOS/VS, or AOS/VS II system, use the DUMP_II utility, not CPIO_VS or TAR_VS.

Upper- and Lowercase Pathnames

In UNIX, filenames are case sensitive; for example, the names myfile and MYFILE indicate different files. Lowercase filenames are customary on UNIX systems. When you use CPIO_VS to dump files, it dumps the names in the case you specify. For example, the command CPIO_VS/OUTPUT...myfile dumps MYFILE with its name in lowercase. If you use a data file built with the !FILENAMES pseudomacro, the filenames will be specified in uppercase (as returned by the CLI from !FILENAMES). To display the case the filenames were dumped under, use the /VERBOSE switch.

Generally, when you are dumping from an AOS/VS, AOS/VS II or UNIX system, there is no password file in :etc:passwd, so the User IDs on files in the dump file will be −1. (On AOS/VS and AOS/VS II systems, the User ID is the username, not a number; on UNIX systems, the path for the file containing User IDs is /etc/passwd.)

When loading, if you specify a template or pathname, the program will look for names with precisely the case you specify. It will convert filenames to uppercase as it loads them. The switches /VERBOSE/TELL show the case of filenames in the dump file without loading the files.

Conversion During Dumping or Loading

While dumping (CPIO_VS/OUTPUT), the utility converts certain characters and identifiers so that the dump file will load correctly on a UNIX system. While loading (CPIO_VS/INPUT), the program converts other characters and identifiers as follows so that the dump file will load correctly on AOS/VS and AOS/VS II.

Pathname Conversion

When CPIO_VS writes a dump file, it converts AOS/VS and AOS/VS II pathname characters to UNIX pathname characters as follows.

AOS/VS Character	UNIX Character	Example
: (directory)	/ (directory)	MYDIR:MYFILE becomes MYDIR/MYFILE
$ (dollars)	_ (underscore)	MY$FILE becomes MY_FILE
? (question mark)	_ (underscore)	MY?FILE becomes MY_FILE

All pathname characters are dumped in uppercase.

When CPIO_VS reads a dump file, it converts UNIX pathname characters to AOS/VS and AOS/VS II pathname characters as follows.

UNIX Character	AOS/VS Character	Example
/ (directory)	: (directory)	MYDIR/MYFILE becomes MYDIR:MYFILE
, (comma)	? (question mark)	MYFILE,01 becomes MYFILE?01
– (hyphen)	? (question mark)	MY–FILE becomes MY?FILE

CPIO_VS (continued)

The operating system converts lowercase characters in pathnames to uppercase; for example, myfile becomes MYFILE.

Group ID (GID) and User ID (UID)

While dumping, CPIO_VS sets the group ID (GID) of each file to 0. If when you dump the file, the owner's User ID (UID) exists in file :etc:passwd, then CPIO_VS will write that User ID to the dump file with the file. If the file owner's User ID is not defined in :etc:passwd, then CPIO_VS will write a −1 to the archive as the ID. (Normally on UNIX systems, user IDs are kept in a file whose path is /etc/passwd. The CPIO_VS equivalent of this path is :etc:passwd, so CPIO_VS searches this path.)

Access Control List (UNIX Permissions)

In a dump, CPIO_VS sets the dump file access control list (ACL) to OWR for the owner and R for other users. When CPIO_VS dumps each file, it converts the ACL to UNIX permissions, so far as possible (it can convert the Owner and Other fields only). For example, if the ACL of a nondirectory file is username,OWARE +,E, CPIO_VS will convert the ACL so that after loading on a UNIX system, its permissions will be – rwx – – – – – x.

Link Files

When CPIO_VS creates a dump file (CPIO_VS/OUTPUT), it resolves links and copies the actual resolution file to the dump file. (CPIO_VS does not support the cpio −l option.)

Time Last Modified and Time Last Accessed

When CPIO_VS creates a dump file, it dumps the existing time last modified and time last accessed along with each file. When the program loads from a dump file (CPIO_VS/INPUT), it changes the time last modified and time last accessed to the time of the load. (DUMP/DUMP_II and LOAD/LOAD_II behave the same way.)

(CPIO_VS does not support the cpio or tar −m switch or the cpio −a switch.)

CPIO_VS Utility Switches

/BLOCK	Tells the program to write data at 10 disk blocks (5,120 bytes) per tape record. By default, it writes only 512 bytes per block. This is analogous to the/BUFFERSIZE=5K switch. The larger block size can speed up I/O and allow more material to fit on tape. This switch is primarily useful for dumps. On a load, CPIO_VS tries to match the block size used for the dump, therefore you can omit this switch.
/DATA=file-with-files	Tells CPIO_VS to read a list of pathnames from file-with-files. You can build these pathnames into the file with the FILESTATUS/L=pathname command. If you omit this switch, the program prompts for each filename (it uses standard input). You can use this for dumps (CPIO_VS/OUTPUT) only.

/DEVICE=dumpfilename	Tells the utility to use dumpfilename as the dump file (for example, @MTB0:0 or MY_DUMPFILE).
/DIRECTORY	Tells the program to create directories as needed. You can use this for loads (CPIO_VS/INPUT) and copies (CPIO_VS/PASS) only. If the dumpfile includes directories and you omit this switch, the directories will be loaded as flat files; the structure on the tape will not be maintained in the working directory.
/F	Tells the program to load all files *except* those specified by pathname. You can use /F only with CPIO_VS/INPUT. It is analogous to the LOAD/LOAD_II backslash pathname template. The program looks for the pathname in precisely the case you specify, unless you use a template character, in which case it looks for the filename/pathname in uppercase.
/RENAME	Tells the program to rename files interactively. It asks whether you want to rename each file. To rename, type the new filename; the program will then load the file under the new name. If you don't want to rename the file, press NEW LINE; the program will then skip (not load) that file. Use /RENAME for loading (CPIO/INPUT or CPIO/PASS) only.
/TELL	Tells the program to display filenames but not load them. Use this with CPIO/INPUT only. It is analogous to the LOAD/LOAD_II switch /N.
/UNCONDITIONAL	Tells the utility to copy files unconditionally. If you omit this switch, the program usually will not replace a newer file with an older file of the same name. You can use this only for loads (CPIO_VS/INPUT) or copies.
/VERBOSE	Tells the program to display the names of files loaded (CPIO_VS/INPUT only). Use this with /TELL to provide a detailed listing.

CPIO_VS Example 1

```
) CPIO_VS/OUTPUT/DEVICE=@MTJ0:0/V↵
myprog.c↵
CTRL–D
myprog.c
21 Blocks
)
```

This sequence shows CPIO_VS dumping a file in interactive mode (run without an argument). The output device for the dump file is the first file of the tape on unit MTJ0. The program waits for a filename; the user enters MYPROG.C; the program waits for another filename; the user signifies the end of the list by pressing CTRL–D; the program then dumps the file, verifies this, and describes the size of the file (21 disk blocks). The file is dumped with its filename in lowercase.

The tape can be taken to a UNIX system for loading via cpio –input. The output file (/DEVICE=) can be a disk file, which can be copied over a network to a UNIX system; then it, too, can be loaded with a cpio –input command.

CPIO_VS Example 2

```
) WRITE/L=DFILE [!FILENAMES MYPROG.C]↵
) CPIO_VS/OUTPUT/V/DEVICE=@MTJ0:0/DATA=DFILE↵
MYPROG.C
21 Blocks
)
```

This sequence produces the same result as the previous one, with the filename specified in a disk file instead of interactively. The WRITE command copies the filename MYPROG.C to file DFILE. CPIO_VS then dumps this file to tape.

CPIO_VS Example 3

```
) CPIO_VS/INPUT/TELL/DEVICE=@MTJ0:0↲
.
(Displays filenames on tape without loading them)
.
) CPIO_VS/INPUT/V/DEVICE=@MTJ0:0↲
.
(Displays filenames loaded)
)
```

This CPIO_VS/INPUT sequence uses the /TELL switch to list filenames in the first file of the tape on unit MTJ0 without loading them; then it loads all those file into the working directory. The dump file on tape was created by the UNIX command cpio –output.

```
) CPIO_VS/INPUT/DIRECTORY/DEVICE=@MTJ0:0  MEMO:AL  MEMO:B+↲
```

This CPIO_VS command loads from the dump file in MTJ0:0. It creates directories as needed in the working directory and loads files that match the templates MEMO:AL and MEMO:B+ into them.

CPUID

Command

Displays the central processing unit identifier.

Format

CPUID

This command displays the identifier for your computer's central processing unit (CPU).

- No arguments.
- Requirement: *Standard.*
- See also: SYSID (to display or set the name of your system.)

Why Use It?

Use the CPUID command if you need to know the identifier for your CPU. This identifier can provide information about your system, such as the CPU model, current microcode revision number, and the amount of memory available on the system. (You can get all this information, except for CPU model, with the SYSINFO command.)

The meaning of the identifier varies with different machine models. Refer to the "Principles of Operation" manual for your system for further information about the CPU identifier.

Command Switches

The section "Universal CLI Switches," earlier in this chapter, describes switches /1, /2, /L, /L=pathname, and /Q, which you can use with all commands. That section also explains the CLI32 switches /STR= and /ESTR=.

CPUID Example

```
) CPUID↵
CPUID 4223002437
```

DISMOUNT *Command*

Asks the system operator to physically dismount a tape.

Format

DISMOUNT linkname *[message]*

This command requests the system operator to remove a tape (identified by the link name assigned in the earlier MOUNT request). You can supply a message that will be passed to the operator.

If necessary, this command rewinds the mounted tape. It also restores the tape unit access control list to what it was before the tape was mounted and you gained exclusive access to the tape unit.

Refer to *Managing AOS/VS and AOS/VS II* for a complete discussion of labeled and unlabeled mount operations, with detailed examples.

- No templates.
- No argument switches.
- Requirement: *Standard.*
- See also: MOUNT.

Why Use It?

Use the DISMOUNT command if you want the system operator to remove a magnetic tape from its unit. You should request a tape dismount as soon as you have finished the tape, so that the unit will be available to others.

Command Switches

The section "Universal CLI Switches," earlier in this chapter, describes switches /1, /2, /L, /L=pathname, and /Q, which you can use with all commands. That section also explains the CLI32 switches /STR= and /ESTR=.

/DIRECTORY=pathname	Uses the link name in the specified directory. If you omit this switch, the system looks for the link name in your working directory. If you used this switch with MOUNT, you must use it with DISMOUNT as well.

DISMOUNT Example

```
) MOUNT/VOLID=BTF008 MYTAPE  Please↵     (System operator mounts the tape.)
) LOAD_II/V  MYTAPE  TAPEFILE:#↵          (System loads the contents of tape.)
.
) DISMOUNT MYTAPE Thanks — Please return tape to rack↵
```

After the system operator mounts the tape that has volume ID BTF008 (which will be called by its link name MYTAPE, the LOAD_II program loads files from the labeled tape file named TAPEFILE (created earlier by DUMP/DUMP_II). The DISMOUNT command asks the operator to remove and store the tape.

DISPLAY

Utility

Displays a file's contents in numeric and ASCII values.

Format

XEQ DISPLAY input–pathname *[destination–pathname]*

This utility displays the contents of a file on your terminal or writes them to a listing file. It can also copy the contents of a file to another file, *destination–pathname*, after converting from EBCDIC to ASCII or otherwise processing them. You can use DISPLAY with any file, including fixed-length record files, EBCDIC files, magnetic tape files, and diskettes.

The default listing format is 16 input characters per line, printed first as octal values and then as text. The ASCII characters that the CLI cannot print as text (those whose octal value is less than 40 and greater than 176_8) print as a period (.). Repeated identical lines print as a series of asterisks (****) .

The default values for the input parameters are FIRST=0, INCREMENT=1, and LAST=32767. If the input is on tape, the tape input block size is the size specified for the device when the system was generated.

If you specify a tape destination pathname, the default values for the output parameters are OBLOCKSIZE= the block size of the input tape; and ODENSITY=0. DISPLAY will create the destination file if it does not already exist. If it does exist, DISPLAY will delete the file and then recreate it.

- Does not accept templates.
- Accepts utility switches (described later).
- Requirement: *Standard* (E access to the program file in :UTIL).
- See also: FILCOM (to compare two binary files), FED disk editor (described in a manual named in the Preface), and BROWSE.

Why Use It?

Use DISPLAY to see the ASCII characters in a file that the TYPE command will not handle properly because the file does not have NEW LINE delimiters. DISPLAY also shows you the numeric values (in octal, decimal, or hexadecimal) for a file's contents, including nonprintable characters. You can use it to examine a program's output file, which can help you trace program operation and detect errors.

The DISPLAY utility can convert EBCDIC text on IBM tapes to ASCII format. It can show you the fields in a tape label. And you can use it to copy one tape to another.

DISPLAY Switches

/8BIT	Displays printable 8-bit characters in the last field. (By default, the system assumes 7-bit characters.)
/ALL	(Only if the input is a tape file) Processes all files to the logical end of tape. Do not specify any tape file number for either input or output (for example, specify @MTB0, not @MTB0:1). If you specify a destination file, and it is a tape, each input file is copied to a separate output file. If you specify a destination file on disk, the first input file will replace the disk file, unless you use /APPEND to add information to an existing file. All other input files are appended to that file.
/APPEND	Appends the output to the destination file if it exists. (If you omit this switch, the destination file will be replaced if it exists.)
/BYTE	Lists the file in byte format (octal values) with no text.
/CONVERT*[=/value]*	Converts EBCDIC input into ASCII. The value can be TEXT (converts text field only), NUMERIC (converts numeric field only), or BOTH (converts both numeric and text fields). The default is text only.
/DECIMAL	Lists the file in decimal instead of octal values.
/FIRST=n	Specifies the first block to be processed, where n can be 0 through 32767. (The default value is 0.)
/HEXADECIMAL	Lists the file in hexadecimal instead of octal values.
/IGNORE	Ignores the logical end of tape on the input file.
/INCREMENT=n	Processes every nth block of the input file.
/IPHYSICALEOT	Ignores physical end of tape (EOT) on the input file and goes to the next EOF.
/L	Appends output to current list file instead of to @OUTPUT.
/L=pathname	Appends output to the file specified by pathname instead of to @OUTPUT.
/LAST=n	Specifies the last block to be processed, where n can be 0 through 32767. (If you use /FIRST, the value of /LAST must be greater than or equal to the value of /FIRST.)
/LISTUDA	Lists the UDA (User Data Area) of the input file.

/NOLIST	Does not display the file contents (useful if you want to convert data only).
/NUMERICONLY	Lists only the numeric value of the input file; omits the text value.
/OBLOCKSIZE=n	Specifies the block size of the destination tape, where n is the number of bytes.
/ODENSITY=value	Specifies the density of the destination tape. Use this switch only if the tape unit supports multiple densities. The following values are valid:

800	(800 bytes per inch)
1600	(1600 bytes per inch)
6250	(6250 bytes per inch)
ADM	(Automatic Density Matching)
LOW	
MEDIUM	(reverts to LOW on a dual-density unit)
HIGH	

If you intend to use a tape on another unit, be careful about choosing LOW, MEDIUM, or HIGH: "low" or "high" on one unit may not be compatible with the values on another unit, which would prevent reading from the tape.

The default is the value selected for the tape unit during VSGEN. If you are using labeled tape or writing to a file other than file 0, the system enforces the density already used on the tape. (If you specify a density that conflicts with the current density, the command fails.)

/RELATIVE	Displays file addresses relative to the first block displayed instead of to the beginning of the file. (This switch is useful with /FIRST.)
/TEXTONLY	Lists text values only; omits numeric values.
/UPPERCASE	In text display, converts lowercase letters to uppercase.
/WIDTH=n	Specifies the number of characters per line in the listing file. The value must be an even number not greater than 24.

DISPLAY Example 1

```
) XEQ DISPLAY/L=@LPT  DATA_89⤶
```

This command prints on the line printer the contents of file DATA_89 in octal and ASCII text values.

DISPLAY Example 2

```
) XEQ DISPLAY/ALL/HEXADECIMAL   @MTD0⤶
```

This command displays the hexadecimal and ASCII text values of all the files on the tape unit MTD0.

DISPLAY Example 3

```
) XEQ DISPLAY/CONVERT  @MTD0:1  GIOTTO⤶
```

This command converts the EBCDIC contents of the first tape file on drive MTD0, and writes the translation to a disk file called GIOTTO.

DISPLAY Example 4

```
) XEQ DISPLAY/CONVERT/NOLIST/ALL  @MTD0  TAPE24⤶
```

This command converts all the files from the tape mounted on unit MTD0 from EBCDIC and copies them to a disk file called TAPE24.

DUMP *Command*

Dumps specified file(s) from the working directory to a dump file (CLI16 only).

Format

DUMP dumpfile *[source−pathname] [...]*

Dumps one or more files from the working directory to the specified dump file. The dump file can be

- a file on a magnetic tape, such as @MTB0:0;
- a tape linkname, such as MYTAPE (after you use the MOUNT command);
- a disk file pathname, such as DUMPDIR:DUMPFILE, which the command will try to create (it cannot already exist);
- a diskette devicename, such as @DPJ10; or
- the generic labeled diskette filename, @LFD (AOS/VS only).

You can specify pathnames (including templates) for files in the working directory. If you omit pathname arguments, the command dumps all files from the working directory and its subordinates; it assumes the template character #. (The /TYPE=\CPD and /TYPE=\DIR switches will *not* exclude directory files from the dump if you use the # template or omit source−pathname). The command retains the directory tree structure within the dump file unless you use the /FLAT switch.

If you intend to dump database files (such as those created and used by INFOS II, DG/SQL, and DG/DBMS software), be sure that the files are properly closed. You may have to run the recommended verification program if an abnormal shutdown occurred.

- Accepts templates (for source−pathname only).
- No argument switches.
- Requirement: *Standard.* You need Read access to all files and Execute access to all directories that you want to dump, or Superuser on.
- See also: DUMP_II, LOAD, LOAD_II, OPERATOR, MOUNT, DISMOUNT.

Why Use It?

Keeping a copy of your data provides backup in case something happens to the disk that holds it, or if you accidentally delete a file you want. Use the DUMP command to transfer files from disk to magnetic tape, disk, or diskettes.

DUMP (continued)

The DUMP_II utility (described next) is superior to the DUMP command for backup because it is faster than DUMP, it lets you dump to multiple unlabeled tape volumes, and it lets you recover from most hard tape errors. However, DUMP_II cannot use labeled diskettes; if you want to use labeled diskettes, you must use the DUMP command. The DUMP command is available from CLI16 only.

Unless you use labeled tape (via MOUNT/VOLID=xxx), the DUMP command cannot dump material to more than one tape volume. If the material you want to dump may require more than one tape, and you don't want to use labeled tape, use the DUMP_II utility, not the DUMP command.

CAUTION: *In rare cases, MTJ-type tape drives, except for the 21-Mbyte cartridge tape, may generate the message* Fatal buffered tape error *during a dump. These drives cannot recover from this kind of error. If you get this error, and you know that the problem is not with the drive, discard the tape and do not reuse it. Restart the dump. See the Notes and Warnings section of the release or update notice for the latest status of this problem.*

To dump files for loading on a UNIX system, use the TAR_VS or CPIO_VS utility.

Command Switches

The section "Universal CLI Switches," earlier in this chapter, describes switches /1, /2, /L, /L=pathname, and /Q, which you can use with all commands. That section also explains the CLI32 switch /STR=.

/AFTER/TLA=date–and/or–time /AFTER/TCR=date–and/or–time /AFTER/TLM=date–and/or–time	Selects files last accessed (/TLA=), created (/TCR=), or last modified (/TLM=) on or after the specified date and time (dd–mon–yy:hh:mm:ss), date (dd–mon–yy), or time (hh:mm:ss). /TCR takes a date–time value with CLI32 only. Seconds and minutes are optional. You can use /BEFORE with /AFTER to specify a span of time.
/BEFORE/TLA=date–and/or–time /BEFORE/TCR=date–and/or–time /BEFORE/TLM=date–and/or–time	Selects files last accessed (/TLA=), created (/TCR=), or last modified (/TLM=) on or before the specified date and time (dd–mon–yy:hh:mm:ss), date (dd–mon–yy), or time (hh:mm:ss). /TCR takes a date–time value with CLI32 only. Seconds and minutes are optional. You can use /AFTER with /BEFORE to specify a span of time.

/SEQUENTIAL	Prevents the program from rewinding the tape after completing the dump. This saves rewind and spool forward time if you want to place another dump file on the same tape volume.
/BUFFERSIZE=n	Sets the I/O buffer size to n bytes, up to the limit set at system generation. ECLIPSE MV/3000 series systems and above allow a maximum of 32768. On all systems, 21-, 120-, 150-, 320-, and 525-Mbyte cartridge tapes allow a maximum of 16384. You can also specify this as mK; for example, 8K means 8192 bytes. A larger size lets more data fit on a tape. We suggest a multiple of 1024 for n; for MTJ units, we suggest 16384. The default buffer size, if you omit this switch, is 2048. The same buffer size you specify here must be specified later when the dump file is loaded.
/DENSITY=value	Specifies the tape density. The value can be 800, 1600, or 6250 bpi (bits per inch); ADM (Automatic Density Matching); LOW, MEDIUM (reverts to LOW on a dual-density unit), or HIGH. You can use this switch with variable density (MTB, MTD) unit types. If you intend to use a tape on another unit, be careful about choosing LOW, MEDIUM, or HIGH: "low" or "high" on one unit may not be compatible with the values on another unit, which would prevent reading from the tape. The default is the value selected for the tape unit during VSGEN. If you are dumping to labeled tape (LABEL utility) or to a tape file other than file 0, the system will enforce the density already recorded on the tape. If you specify a density that conflicts with the density recorded on the tape, the command will fail.
/FLAT	Dumps all files, including subordinate directories, as if they were nondirectory files; eliminates the directory structure.
/IBM	Dumps to a tape with an IBM–format label. The label must have been written via the LABEL utility with the /I switch. The program will write the data itself (not the labels) in AOS/VS dumpfile format, not IBM format. To read this tape, use LOAD with the /IBM switch.
/NACL	Does not dump file access control lists (ACLs) with files. When loaded, files will get the default ACL of the process that loads them.

DUMP (continued)

/RETAIN=ndays	Sets the retention period on a labeled tape or diskette to ndays. The file cannot be overwritten until the retention period expires (unless someone relabels the tape with the LABEL program). The default retention period is 90 days. This switch applies only to labeled tape and labeled diskettes.
/SEQUENTIAL	Prevents the program from rewinding the tape after completing the dump. This saves rewind and spool forward time if you want to place another dump file on the same tape volume.
/SPECIFIC	Dumps to the tape volume specified as the dump file (you must use an implicit mount via @LMT:volid:tape–filename). Use this switch to add a logical file to the end of a file set without going through all preceding volumes. If you use DUMP/SPECIFIC to start a new logical file, use LOAD/SPECIFIC to load from that logical file. This switch is further described in Chapter 6.
/TYPE=typecode	Dumps files of type code only, where the typecode variable is one of the codes shown in Table 2–8. You can use the following forms for type code.

xxx	a 3–letter mnemonic (such as DIR or CPD for a directory, LNK for link).
n	a decimal number (0–255) that defines a type code.
m–n	decimal numbers that select a range of type codes.

/TYPE=\typecode	Dumps files except those of type code, where type code is one of the code types listed in the left column above. You can use more than one /TYPE= switch in a command.
/V	Displays the name of each file that is dumped.

DUMP Example 1

```
) DUMP/V  @MTB0:0 ⤶

.
(CLI displays list of files dumped)
.
```

@MTB0 is the name of the magnetic tape unit. The 0 indicates the first tape file. The CLI dumps all files from the working directory, including subordinate directories and their files, into file 0, maintaining the directory tree structure.

DUMP Example 2

```
) DUMP/V/NACL  @MTD0:1 ⤶

.
(CLI displays list of files dumped)
.
```

This command dumps the contents of the working directory and all subordinate directories (because no source pathname was given) to the second file of the magnetic tape mounted on MTD unit 0. The files are dumped without ACLs (so that when they are loaded, they will be given the current default ACL). The /V switch displays each file's pathname as it is dumped.

DUMP Example 3

```
) DUMP/V/L=SOURCES.23.MAY.90/AFTER/TLM=23–MAY–90  @MTB0:1  & ⤶
&) F77:–.F77  MASM:–.SR ⤶
```

This dumps FORTRAN 77 and assembly language source files in the directories F77 and MASM (which are in the working directory) that were created or last modified after May 23, 1990. It dumps them into the second tape file of the tape on unit 0. And it writes the listing of dumped files to file SOURCES.23.MAY.90.

DUMP Example 4

```
) DUMP/V/NACL :UDD:COMMON:ARCHIVES:CDFDUMPFILE  +.CDF ⤶
```

This command dumps all files in the working directory ending with the .CDF filename suffix to a disk file called CDFDUMPFILE in the directory :UDD:COMMON:ARCHIVES. The /V switch displays each file's pathname as it is dumped; the /NACL switch dumps the files without ACLs.

DUMP Example 5

```
) SUPERUSER ON↵
Su) MOUNT/VOLID=V1/VOLID=V2/VOLID=V3 MYTAPE Ready for backup↵

... (System operator mounts tapes.)

Su) DIR :↵

Su) DUMP/BUFFERSIZE=16384/V/L=:UDD:[!USERNAME]:SYSTEM_BACKUP &↵
&Su)   :UDD:[!USERNAME]:MYTAPE:FILESET1↵

... (System dump occurs to multiple volumes.)

Su) DISMOUNT MYTAPE Backup is done.↵
```

This example shows a system backup, with a multiple volume labeled tape dump. The MOUNT command asks the system operator (person at the system console) to mount tape volume V1 (first of a sequence of three volumes). The DIRECTORY command makes the root directory the working directory. The DUMP command then dumps all files in the system, maintaining directory structure, to tape file FILESET1 using the name MYTAPE (linkname MYTAPE is created in the initial user directory). The dump can use up to three tape volumes. The dump listing goes to file SYSTEM_BACKUP in the person's initial user directory. After the dump is done, the person types DISMOUNT, prompting the operator to dismount the tape(s).

DUMP_II

Utility

Dumps one or more files from the working directory to the specified dump file.

Format

DUMP_II dumpfile *[source–pathname] [...]*

The DUMP_II utility creates dump files that provide backup for your disk–based information. Later, you can load these dump files with the LOAD_II utility or the CLI command LOAD. The DUMP_II utility dumps files from the working directory to the dump file. The dump file can be one of the following.

- a file on magnetic tape, such as @MTB0:0
- a tape link name, such as MYTAPE (after you use the MOUNT command)
- a disk file, such as DUMPDIR:DUMPFILE, which the program will create
- a device name, such as @DPJ10 (but not the labeled diskette file, @LFD).

You can specify pathnames (and templates) for files in the working directory. If you omit pathname arguments, the utility dumps all files in and below the working directory; it assumes the template character #.

NOTE: If you intend to dump database files (such as those created and used by INFOS II, DG/SQL, and DG/DBMS software), be sure that the files are properly closed. You may also have to run the recommended verification program if an abnormal shutdown occurred.

Under AOS/VS II only, you can dump a user-defined system area in the range 1001–9999. You might need to do this if you have enabled automatic dumping of memory dumps to a system area, and later want to transfer the system area to tape. For example,

Su) DUMP_II @MTD0:0 @DPJ0:2001 ⤶

dumps system area 2001 on disk DPJ0 to tape file 0 of the tape mounted on the first MTD drive. DUMP_II changes the name of the system area by replacing the @ with a ? and the : with an _ character. In this example, the system area would become filename ?DPJ0_2001.

DUMP_II (continued)

The utility copies the directory tree structure to the dump file unless you use the /FLAT switch to suppress the directory structure.

DUMP_II dumps multiple files in the order they are found in the directory, which usually differ from the order they appear on the command line.

Most tape units check the readability of data as they write it. However, if you want to verify that the dump file is loadable, use the LOAD_II/N command after the dump completes to read the dump file without loading files.

DUMP_II and a macro to execute it are supplied in the root directory. In addition, a link to the utility and a copy of the macro to execute it are supplied in directory :UTIL.

- Accepts template for pathnames and filenames.
- Accepts utility switches (described later).
- Requirement: *Standard* (Execute access to the program file in :UTIL or the root). You need Read access to all files and Execute access to all directories that you want to dump, or Superuser on.
- See also: MOUNT, LABEL, LOAD_II.

Why Use It?

Keeping a copy of your data provides backup in case something happens to the disk that holds it, or if you accidentally delete a file you want.

The DUMP_II utility is superior to the DUMP command for backup because it is faster than DUMP, it lets you dump to multiple unlabeled tape volumes, and it lets you recover from most hard tape errors. However, DUMP_II cannot use labeled diskettes; if you want to use labeled diskettes, you must use the DUMP command. The DUMP command is available from CLI16 only.

CAUTION: *In rare cases, MTJ-type tape drives, except for the 21-Mbyte cartridge tape, may generate the message* Fatal buffered tape error *during a dump.*

These drives cannot recover from this kind of error. If you get this error, and you know that the problem is not with the drive, discard the tape and do not reuse it. Restart the dump. See the Notes and Warnings section of the release or update notice for the latest status of this problem.

To dump files for loading on a UNIX system, use the TAR_VS or CPIO_VS utility.

Multiple-Volume Dumps

The DUMP_II (and LOAD_II) programs can use multiple-volume dump files without using labeled tape. For example, if you type

```
) DUMP_II  @MTB0:0⤶
```

and all files in the working directory won't fit on the tape volume on tape unit MTB0, the utility will prompt for any additional volumes by displaying

Mount the next volume, volume: n
Respond MOUNTED <device> or REFUSED when ready.

To continue the dump, you can then mount another tape — on the same unit or on a different unit. If you use the same unit, you can simply type

```
) MOUNTED⤶
```

or an abbreviation, omitting a device name. If you use a different unit, you must also type the unit name. (With unlabeled tape, the utility cannot check the volume mounted to make sure it is not the tape just written to, so don't type MOUNTED until you've actually mounted another tape.)

Note that when DUMP_II continues a dump on a new tape, the continuation begins at tape file 0 on the new tape; if you issue another DUMP_II command to this volume, you must specify tape file 1. For example, you issue three dump commands (DUMP_II ... @MTD0:0, DUMP_II ... @MTD0:1, and DUMP_II ... @MTD0:2), and the third dump continues to another tape; when the third dump completes, you must specify file 1 in the next dump command (*not* file 3), since the dump continuation started at file 0 on the new tape.
In batch mode, DUMP_II always begins by trying to create an IPC file in your working directory. It might need this file later to communicate with the system console. So, you should have Write access to your working directory or have Superuser privilege turned on. If DUMP_II cannot create this IPC file initially, it will continue running. However, if it needs to communicate later with the system console, the utility will abort.

DUMP_II communicates with the system console in batch mode when the dump requires more than one reel of unlabeled tape, and for error conditions. DUMP_II then sends an informative message to the system console and awaits a response. The person at the system console normally responds with a message of the form

```
CONTROL  :UDD:username:DUMPpid MOUNTED tape–unitname
```

to continue the dump. Or, the person at the system console can refuse to mount another reel, thus ending the dump operation abnormally.

DUMP_II (continued)

For example, suppose you are user LISA and execute DUMP_II in batch mode. Furthermore, suppose DUMP_II runs as PID 77 and has filled the first reel of unlabeled tape on unit @MTB2. It displays the following on the system console.

```
From Pid 77:(DUMP00077)    The tape is rewinding ...
From Pid 77:(DUMP00077)    Mount the next volume, volume: 02
From Pid 77:(DUMP00077)    Respond CONTROL :UDD:LISA:LOAD00071
From Pid 77:(DUMP00077)    Respond MOUNTED <device> or REFUSED when ready.
```

To continue the dump, you (or the operator) change tape reels and respond with

```
) CONTROL :UDD:LISA:DUMP00077 MOUNTED @MTB2)
```

To stop the dump, you (or the operator) respond with

```
) CONTROL :UDD:LISA:DUMP00077 REFUSED)
```

DUMP_II Switches

The section "Universal CLI Switches," earlier in this chapter, describes switches /1, /2, /L, /L=pathname, and /Q, which you can use with all commands. Switches /1= and /2= must occur first on the command line. Otherwise, error handling may not occur properly.

/AFTER/TLA=date–and/or–time /AFTER/TCR=date–and/or–time /AFTER/TLM=date–and/or–time	Selects files last accessed (/TLA=), created (/TCR=), or last modified (/TLM=) on or after the specified date and time (dd–mon–yy:hh:mm:ss), date (dd–mon–yy), or time (hh:mm:ss). Seconds and minutes are optional. You can use /BEFORE with /AFTER to specify a span of time.
/BEFORE/TLA=date–and/or–time /BEFORE/TCR=date–and/or–time /BEFORE/TLM=date–and/or–time	Selects files last accessed (/TLA=), created (/TCR=), or last modified (/TLM=) on or before the specified date and time (dd–mon–yy:hh:mm:ss), date (dd–mon–yy), or time (hh:mm:ss). Seconds and minutes are optional. You can use /AFTER with /BEFORE to specify a span of time.

/BLOCKCOUNT=n	Specifies the number of blocks DUMP_II will write to the dump medium in one output operation. A block equals the buffer size (if dumping to tape) or 512 bytes (if dumping to disk). Values range from 1 block (the default) to 255. If you use 255 blocks, the utility will write as many blocks as will fit in its buffers, and may therefore improve processing when you are dumping to a disk file. When you are dumping small files to tape, use this switch in conjunction with a small buffer size (such as 2048, the default) to improve performance.
/BOTH*[=pathname]* or /B*[=pathname]*	Lists information both to your terminal and file pathname. Omit pathname to use the generic @LIST file. If pathname doesn't exist, DUMP_II creates it; if it does exist, DUMP_II appends to it. (To display filenames, you must also use the /VERBOSE switch, which you can abbreviate to /V.)
/BUFFERSIZE=n	Sets the I/O buffer size to n bytes, up to the limit set at system generation. ECLIPSE MV/3000 series systems and above allow a maximum of 32768. On all systems, 21-, 120-, 150-, 320-, and 525-Mbyte cartridge tapes allow a maximum of 16384. A larger size lets more data fit on a tape and usually improves performance. Use an even multiple of 1024. The default buffer size, if you omit this switch, is 2048 bytes.
/DENSITY=value	Specifies tape density. The value can be 800, 1600, or 6250 bits per inch; ADM (Automatic Density Matching); LOW, MEDIUM (reverts to LOW on a dual-density unit), or HIGH. You can use this switch with variable density (MTB, MTD) unit types. If you intend to use a tape on another unit, be careful about choosing LOW, MEDIUM, or HIGH: "low" or "high" on one unit may not be compatible with the values on another unit, which would prevent reading from the tape. The default is the value selected for the tape unit during VSGEN. If you are dumping to labeled tape (LABEL utility) or to a tape file other than file 0, the system enforces the density already recorded on the tape. If you specify a density that conflicts with the density recorded on the tape, the command fails.

/ERROR=pathname or /E=pathname	Writes error messages to the specified file. Error messages also go to the terminal or batch output file, and to the listing file. The file will be created if it does not exist; if it does exist, text will append to it.
/FASTFORWARD	Speeds up positioning an unlabeled tape on QIC, 4-mm DAT, and 8-mm cartridge tapes. (Other tape drives implement fast forward in hardware, so this switch has no effect with them.) Use this switch when specifying a large tape file number (for example, @MTJ0:150), which might otherwise cause a device time-out. You do not need to use the /MAXCAPACITY switch when using this switch; selecting this switch also sets maxcapacity mode. This switch ignores logical end-of-tape marks. As a result, if you specify a tape file number that does not actually exist, you receive an error when the tape positions beyond the last file on the tape. The systems that support this switch are ECLIPSE MV/4000® and above, excluding ECLIPSE MV/5000™ DC series systems.
/FLAT	Dumps subordinate directories as if they were nondirectory files; eliminates directory structure.
/IBM	Dumps to a tape with an IBM–format label. The label could have been written via the LABEL utility with the /I switch. The program will write the data itself (not the labels) in AOS/VS dumpfile format, not IBM format. To read this tape, use LOAD_II or the LOAD command with the /IBM switch.
/MAXCAPACITY	Runs tape unit in streaming and buffered mode (useful only with some cartridge units). Streaming can increase tape capacity and speed, but if the unit tries to stream and fails, you'll hear it starting and stopping; I/O will be slower than without this switch. Use this switch for large backup operations, when the system is otherwise idle.
/MEMORY=value	The switch has no effect. To maintain compatibility with AOS/VS Revision 7.63 and earlier, the switch accepts the following values: an integer 1 through 200 or the word MIN, LOW, MED, HIGH, or MAX.

/NACL	Does not dump file access control lists (ACLs) with files. When loaded, files will get the default ACL of the process that loads them.
/NCOMPRESS	Hardware data compression is the default on ultra-high capacity cartridge tapes. Use this switch to disable hardware data compression only when you want to dump to a tape for transfer to a unit that cannot read compressed data.
/NPERMANENCE	Does not dump the permanence attribute with files. When loaded, all files will have permanence off.
/NPROMPT	Terminates the dump operation if the utility encounters an error that requires interaction. If an error occurs that normally produce an interactive prompt, the utility terminates and forces you to restart the dump rather than accepting any recovery action. (EXEC still allows normal operator intervention with labeled tapes.) This switch is useful to ensure an absolutely error-free dump.
/NSPAN	(Unlabeled tapes only.) Does not span more than one reel of tape. If you try to place more data on a reel of tape than it will accept, DUMP_II terminates with an error message.
/OWNER=name	Verifies that the tape belongs to the owner you specify with name. The utility terminates with an error message when the name field from the tape does not match the specified name.
/READCOUNT=n	Specifies the number of disk blocks DUMP_II will read from each file on one input request. The range is 1 to 255. If you omit this switch, the default is 32 blocks.
/RETAIN=n	Sets the retention period on a labeled tape to n days. The file cannot be overwritten until the retention period expires (unless someone re-labels the tape via the LABEL program). The default retention period is 90 days.
/SEQUENTIAL	For labeled tape that has been mounted via MOUNT/VOLID= and the EXEC command MOUNTED. Prevents the program from rewinding the tape after completing the dump. This saves rewind and spool forward time if you want to place another dump file on the same tape volume.

/SPECIFIC	Dumps to the specified labeled tape volume. The utility could reference that volume via @LMT:volid:tape–filename. Use this switch to avoid processing all the previous volumes from the file set. This switch is further described in Chapter 6.
/STATISTICS	Writes dump statistics to your terminal, or to the listing file if you specify one.

/TRAVERSE=directory–typecode

Specifies the directory types to traverse (go through) while searching for files to dump. Directory typecodes include LDU (logical disk unit), CPD (control point directory), and DIR (standard directory). The utility finds files that exist only in the specified directory types.

/TRAVERSE=\directory–typecode

Traverses all directories except those of directory–typecode. Directory typecodes include LDU (logical disk unit), CPD (control point directory), and DIR (standard directory). The utility finds files that exist only in directory types not excluded with this switch.

/TYPE=typecode or /T=typecode — Dumps files of a certain type code only, where the typecode is one of the following:

xxx	a 3-letter mnemonic (such as DIR or CPD for a directory, LNK for link), listed in Table 2–8.
n	a decimal number (0–255) that defines a type code. Valid file types are system–defined types 0, 10–13, 64–78, 81–103, and 105–127, or user-defined types 128–255.
m–n	decimal numbers that select a range of type codes.

/TYPE=\typecode or /T=\typecode — Dumps files *except* those of the specified type code, where typecode is one of the code types listed in the left column above.

You can use more than one /TYPE= switch in a command.

/VERBOSE or /V — Displays the names of file(s) dumped on your terminal screen, or if you also include the /LISTING=pathname switch, to the specified file. To display names *and* write them to the listing file, also use the /BOTH= switch.

DUMP_II Example 1

```
) DUMP_II/V @MTD0:0↵
.
(Utility displays list of files dumped)
.
```

@MTD0 is the name of the magnetic tape unit. The 0 indicates the first tape file. The program dumps all files from the working directory, including subordinate directories and their files, into file 0, maintaining the directory tree structure. It dumps all files because the command line did not specify a source–pathname. The /V switch displays each pathname as it is dumped.

In this example, and all others following, if more tape is needed, the program will ask for it by displaying the prompt

Mount the next volume, volume: n
Respond MOUNTED <device> or REFUSED when ready.

The person who issued the command can then mount another tape and type MOUNTED or, if using a different unit, MOUNTED unitname. (The utility always prompts for a volume number (n) with unlabeled tape. It cannot check the volume mounted, so don't confirm by typing MOUNTED until you've actually mounted another tape.)

DUMP_II Example 2

```
) DUMP_II/BUFFER=8192/DENSITY=1600/BOTH=DUMP.LIST/V @MTB0:0↵
... (Utility displays list of files dumped) ...
```

This command dumps all files in and beneath the working directory to file 0 (the first file) of the tape on unit MTB0. The buffer size of 8192 uses less tape than the default size of 2048. The /DENSITY switch specifies 1600 bytes per inch for the dump. The switches /BOTH=DUMP.LIST and /V send a listing of all files dumped to the terminal and to file DUMP.LIST.

DUMP_II Example 3

```
) DUMP_II/BUFFER=16384/MAXCAPACITY/BOTH=DUMP.LIST/V @MTJ0:0↵
... (Utility displays list of files dumped) ...
```

This command has the same effect as the previous one, but the switches let the program run the cartridge tape on MTJ0 more efficiently.

EXECUTE (continued)

CLI32 does not require EXECUTE or XEQ to execute a program, but using one of these commands tells the CLI to execute the program directly; otherwise the CLI will look for a macro file before it looks for a program file.

Command Switches

The section "Universal CLI Switches," earlier in this chapter, describes switches /1, /2, /L, /L=pathname, and /Q, which you can use with all commands. That section also explains the CLI32 switches /STR= and /ESTR=.

/I	Creates input for the program from @INPUT. Input cannot contain pseudomacros; the CLI will not interpret them. The last line of input must be a single right parenthesis,).
/INPUT	(CLI32 only.) Same as /I.
/M	Creates input for the program from the macro body. Macro input cannot contain pseudomacros; the CLI will not interpret them. The last line of the macro body unit must contain a single right parenthesis,).
/MACRO	(CLI32 only.) Same as /M.
/S	Stores the program termination message in the current CLI String (instead of sending it to @OUTPUT).
/STRING	(CLI32 only.) Same as /S.

EXECUTE Example 1

```
) EXECUTE  SED  REPORT1↵
```

This command executes the SED text editor program, SED.PR. The argument REPORT1 specifies the file that SED will open for editing. After the editing session is complete and the utility terminates, the CLI prompt returns.

EXECUTE Example 2

```
) EXECUTE MYPROG  0  1↵
```

This command executes a program named MYPROG. The arguments 0 and 1 are available to the program to use appropriately.

EXECUTE Example 3

```
) EXECUTE/S  PROG2↵
```

This command executes a program named PROG2. The program's termination message (if any) is sent to the CLI String rather than to @OUTPUT.

!EXIT *Pseudomacro*

Terminates the current macro or the current macro and any calling macros, or skips to the command following the end of a CLI32 loop sequence (CLI32 only).

Format

[!EXIT]

When executed without a switch, the !EXIT pseudomacro terminates the current macro and any calling macros, returning the user to the CLI32 prompt. Command switches let !EXIT skip to the command following the end of a CLI32 loop sequence, to the next command in the macro that called the currently executing macro, or to the command line.

- No templates.
- No argument switches.
- Accepts macro name switches (described later).
- Requirement: *Standard.*
- See also: !LOOPSTART, !LOOPEND

Why Use It?

Use the !EXIT pseudomacro to provide a conditional exit from a macro or loop.

Macro Name Switches

/LOOP	Redirects execution from a CLI32 loop (introduced by !LOOPSTART) to the statement following !LOOPEND. The loop stops executing whether or not its iteration count limit has been reached.
/MACRO	Skips to the end of the current macro. If this macro was called from another macro (nested), the CLI executes the next command in the calling macro. When the macro is not nested, it returns to the CLI.

!EXIT Example 1

The following macro named MAKESTRINGS.CLI executes a loop a maximum of five times – fewer if the exit condition is met.

!EXIT Example 1 (continued)

```
\\ This macro accepts as many as five file names – storing each
\\ one in a separate CLI string

[!LOOPSTART 5]
STRING/NAME=[!VAR0] [!READ Type name or none and Enter: ]
     [!EQUAL [!STRING/NAME=[!VAR0]],none
            STRING/NAME=[!VAR0]/KILL
            [!EXIT/LOOP]
     [!END]
WRITE String [!VAR0] contains "[!STRING/NAME=[!VAR0]]"
VAR0 [!UADD [!VAR0] 1]
[!LOOPEND]
WRITE Done – [!VAR0] names entered
```

```
) MAKESTRINGS ↵
Type name or none and Enter: FILE1 ↵
String 0 contains "FILE1"
Type name or none and Enter: FILE2 ↵
String 1 contains "FILE2"
Type name or none and Enter: FILE3 ↵
String 2 contains "FILE3"
Type name or none and Enter: NONE ↵
Done – 3 names entered
```

!EXIT Example 2

The following macro named LISTSTRINGS.CLI executes a loop as long as it finds existing string variables.

```
\\ This macro lists as many as five previously defined strings
\\ with their contents.

VAR0       0
[!LOOPSTART ]                          \\ Loop indefinitely
     [!EQUAL [!LENGTH [!STRING/NAME=[!VAR0]]], 0]
            [!EXIT/LOOP]               \\ No more strings defined
     [!END]
WRITE String [!VAR0] contains "[!STRING/NAME=[!VAR0]]"
VAR0 [!UADD [!VAR0] 1]                 \\ Increment string count
[!LOOPEND]
WRITE Done – [!VAR0] strings listed
```

```
) LISTSTRINGS ↵
String 0 contains "FILE1"
String 1 contains "FILE2"
String 2 contains "FILE3"
Done – 3 strings listed
```

!EXIT Example 3

The following macros named CALLING.CLI and CALLED.CLI illustrate the use of !EXIT with no switch and with the /MACRO switch. The called macro uses !EXIT/MACRO to omit processing and return to the calling macro if the file is a link. When the called macro detects a null filename, it executes !EXIT to return to the CLI prompt. The calling macro uses !EXIT to return to the CLI prompt if the specified directory is empty.

```
\\ CALLING.CLI                  Include a pathname argument to list files
\\                              in directory other than the working directory
VAR/NAME=arg 1                                      \\ Initialize the filename counter
PUSH
DIR %1%         \\ In new environment optionally change to directory to be listed

STRING/NAME=FILE [!FILENAMES/NOEQUAL/SORT]      \\ Store filenames
[!EQUAL,([!STRING/NAME=FILE]),()]               \\ Check for null string
    POP                                         \\ To previous environment
    [!EXIT]                                     \\ Return to CLI32 prompt
[!END]

[!LOOPSTART]                                    \\ Start infinite loop
STRING/NAME=ARG [!ARG/ITEM=[!VAR/NAME=ARG] [!STRING/NAME=FILE]]
                                                \\ Store first filename
CALLED [!STRING/NAME=ARG]                       \\ Call macro and pass it list
VAR/NAME=ARG [!UADD,[!VAR/NAME=ARG],1]          \\ Step filename counter
[!LOOPEND]


\\ CALLED.CLI
\\
[!EQ,(%1%),()]                                  \\ Test for null argument
    POP                                         \\ To previous environment
    [!EXIT]                                     \\ Return to CLI32 prompt
[!END]
FILESTATUS/ASSORTMENT/SORT/STR=STATUS %1%       \\ Store status
ACL/STR=ACCESS %1%                              \\ Store access control list

[!NEQ,([!ARG/ITEM=4 [!STRING/NAME=STATUS]]),(LNK)]  \\ Not a LINK
    WRITE [!STRING/NAME=STATUS] [!STRING/NAME=ACCESS]
                                                \\ Write file status and ACL
[!ELSE]
    [!EXIT/MACRO]                               \\ Return to calling macro
[!END]
```

!EXIT Example 3 (continued)

```
) DIRECTORY :TEST ⤶
:TEST
) FILESTATUS/ASSORTMENT/SORTED ⤶

Directory :TEST

BASIC.CLI          TXT   23-Apr-86      8:05:08      68
BCAST.CLI          LNK   BROADCAST.CLI
BROADCAST.CLI      UDF   23-Apr-86      8:05:08     188
SETUP.CLI          TXT   13-Mar-92      8:36:30      15

) CALLING :TEST⤶
Directory:TEST BASIC.CLI TXT    23-Apr-86 8:05:08     68     OP OWARE + RE
Directory:TEST BROADCAST.CLI UDF    12-FEB-90 12:25:36 188    + RE
Directory:TEST SETUP.CLI TXT    13-Mar-92 17:41:58    15     FRAN OWARE $+
+ RE
```

!EXPLODE *Pseudomacro*

Expands each argument into a string of single–character arguments.

Format

[!EXPLODE *[argument] [...]*]

This pseudomacro converts the argument(s) to a single character string, changes delimiter characters to spaces, inserts spaces between the single characters, and then displays each character in the string. Without arguments, the pseudomacro returns a null string.

- No templates.
- No argument switches.
- No macro name switches.
- Requirement: *Standard.*

Why Use It?

Use the !EXPLODE pseudomacro to convert one or more arguments into individual characters. This pseudomacro is most useful for passing arguments from one macro to another.

!EXPLODE Example 1

```
) WRITE [!EXPLODE  ABC]↲
A B C
```

This command separates the pseudomacro argument into its individual characters and displays them.

```
) WRITE [!EXPLODE  ABC]↲
A□B□C
```

Had you used commas to separate the argument characters, as in

```
) WRITE [!EXPLODE  A,B,C]↲
```

the pseudomacro would replace the commas with spaces and add a space to separate each character, resulting in:

```
A□□□B□□□C
```

Why Use It?

Default print specifications (such as lines per page and characters per line) for each printer are based on default values set up by the EXEC program.

Sometimes, you may want to print text using different specifications (for example, to print fewer or more lines per page; many mailing labels are 1 inch high). The FCU utility allows you to set new forms specifications for printing by creating or editing a file's UDA. The file itself need not contain other material — it can be empty, or it can contain text for printing.

Procedure

When the FCU utility starts up, it displays a message like this:

AOS/VS Forms Control Utility Revision xx.xx   *date*   *time*
Type 'Help' for instructions

Command?

You can now type any of the following FCU commands followed by a NEW LINE:

Command	**Meaning**
B	Terminates the Forms Control Utility.
C	Creates forms control specifications for an existing file. If the file already has specifications, use E instead.
E	Edits forms control specifications for a file.
H	Displays all FCU commands.
L	Prints a file's forms specifications to the current list file. Please note that if you use the L command, you must have previously set a list file or have executed FCU with the /L= switch (see FCU switches).
T	Type forms control specifications on the terminal.

If you enter the C or E command, the FCU returns with an interactive question/answer dialog. Default values or current settings are enclosed in square brackets and you may select them simply by pressing NEW LINE. When you change certain parameters, the system gives dependent parameters default values. If you change the line length, for example, the FCU sets default tab stops.

In all, there are 10 questions you must answer to create or edit forms control specifications. A caret (^), produced by pressing Shift–6, moves you back to the previous question and a NEW LINE moves you to the next question. You may use a CTRL–C CTRL–A to interrupt the dialog and return to the *Command?* prompt. (The file's UDA will then be left with default form specifications.) The questions are as follows.

1. *Command?*

   If the file has no forms control specifications, type C. If you want to check or edit existing specifications, type E.

2. *Pathname?*

   Type the pathname or filename of the file whose forms specifications you are creating or editing.

3. *Characters per line (16–255)*
   [80] ?

   Type the maximum number of characters you want on each line. If you press NEW LINE only, you select the default value of 80 characters per line (shown in the square brackets).

   NOTE: At print time, this number must be less than or equal to the line length of the form in the printer.

4. *Tab stops (2–79, OR STANDARD)*
   [8,16,24,32,40,48,56,64,72]
   ?

   To set default tab stops at every eighth column (beginning at column 8), type STANDARD (or an abbreviation of it) or press NEW LINE. Printing begins on the column following the tab stop. Column 1 and the last column are not valid tab stop positions.

5. *Form length in Lines Per Page (6–144)*
   [66] ?

   Type the line number of the last line you want printed on the page before the printer starts a new form. For legible printing, this number cannot be greater than the number of lines on the form.

6. *Top of Form (Channel 1) Line Number (1–lastline)*
   [4] ?

   Type the line number on which you want printing to begin. This number also becomes the setting for channel 1 of the VFU. The parentheses will show the number of the first line and the number of the last line you specified for printing (in question 5).

/L=pathname	Writes output to the file specified by pathname instead of to @OUTPUT.
/RADIX=n	Uses the specified radix when displaying the file contents; only values 2 through 16 inclusive are valid. The default radix is 8.
/UDA_ONLY	Compares the files' User Data Areas (UDA) only, not the file contents.

FILCOM Example

Assume the following three files are in the working directory:

FILE1 contains	FILE2 contains	FILE3 contains
ABCDEFG	ABCDEFG	ABCDEFG
HIJKLMN	ABCDEFG	HIJKLM
OPQRSTU	OPQRSTU	OPQRSTU
		VWXYZ

```
) XEQ FILCOM FILE1 FILE2↵

FILCOM, FILE1, FILE2

               4                44111              40502
               5                45113              41504
               6                46115              42506
               7                47012              43412

Number of words (in decimal) that differed =    4

) XEQ FILCOM FILE1 FILE3↵

FILCOM, FILE1, FILE3

               7                47012               5117
              10                47520              50121
              11                50522              51123
              12                51524              52125
              13                52412               5126
              14  ----------------                 53530
              15  ----------------                 54532
              16  ----------------          <  12>------

Number of words (in decimal) that differed =     8
```

First, the utility compares FILE1 and FILE2, then it compares FILE1 and FILE3. Note that two characters are packed into each word and that the word number in the left column is in octal.

!FILENAMES *Pseudomacro*

Expands to one or more filenames.

Format

[!FILENAMES *[pathname ...]*]

This pseudomacro returns a list of filenames. If you supply an argument, the pseudomacro lists the files that correspond to that filename or template. If you do not supply an argument, the pseudomacro returns a list of the files in the working directory.

!FILENAMES does not resolve links. If you use a link file as an argument to the pseudomacro, the link filename is returned, not the file it resolves to. If the argument contains a link as part of a pathname, the file will not be found, because the pseudomacro does not resolve the link.

NOTE: If you use a template that matches a complex directory structure, the command may overflow memory. To avoid an overflow, execute !FILENAMES from Level 0 with as short a command line as possible.

- Accepts templates.
- No argument switches.
- Accepts macro name switches (described later).
- Requirement: *Standard.*

Why Use It?

Use the !FILENAMES pseudomacro to provide a list of files as an argument to a CLI command or macro.

Macro Name Switches

/AFTER/TLA=date–and/or–time /AFTER/TCR=date–and/or–time /AFTER/TLM=date–and/or–time	Selects files last accessed (/TLA=), created (/TCR=), or last modified (/TLM=) on or after the specified date and time (dd–mon–yy:hh:mm:ss), date (dd–mon–yy), or time (hh:mm:ss). /TCR takes a date–time value with CLI32 only. Seconds and minutes are optional. You can use /BEFORE with /AFTER to specify a span of time.
/BEFORE/TLA=date–and/or–time /BEFORE/TCR=date–and/or–time /BEFORE/TLM=date–and/or–time	Selects files last accessed (/TLA=), created (/TCR=), or last modified (/TLM=) on or before the specified date and time (dd–mon–yy:hh:mm:ss), date (dd–mon–yy), or time (hh:mm:ss). /TCR takes a date–time value with CLI32 only. Seconds and minutes are optional. You can use /AFTER with /BEFORE to specify a span of time.

/COUNT	(CLI32 only.) Counts the number of files listed and displays the count (not the filenames). Use this switch alone; it has no effect together with other switches.
/EXISTS	(CLI32 only.) Tells the CLI to check for the existence of matching files. If the CLI finds one or more matching files, it displays *Yes*; otherwise, it displays *No*. You can place the result in the string and test for Yes or No (see Example 5). Using /EXISTS is faster than counting (/COUNT) and checking for 0. Note that /EXISTS returns the value *Yes* or *No* only; it does not list filenames, even in combination with other switches.
/NOEQUAL	(CLI32 only). Tells the CLI to omit the equals sign (=) before each filename; by default the CLI displays = to indicate the working directory.
/SORT	(CLI32 only.) Sorts the file names alphabetically.
/SORT=	(CLI32 only.) Sorts display according to the key given:

BLOCK	— Number of disk blocks currently used
DCR	— Date of file creation
LENGTH	— File size (bytes)
NAME	— Filenames sorted alphabetically
OPENCOUNT	— ?OPEN count
PERMANENCE	— Permanence on file
TCR	— Date and Time of file creation
TLA	— Date and Time of last file access
TLM	— Date and Time of last file modification
TYPE	— File type

Use a minus (−) prefix to the key to sort in reverse order; and either a plus (+) prefix or no prefix to sort in ascending order. The /SORT= switch may occur up to 12 times on the command line, provided that no sort key repeats. The order in which the keys appear on the line indicates the order of precedence used in sorting the display. The first key is the primary sort key, the second is the secondary sort key, and so on. For example,

```
) WRITE [!FILENAMES/SORT=DCR/SORT=–LENGTH] ⤶
```

displays file names in ascending order of creation date (earliest first) and, within groups having the same creation date, in reverse order of size (largest first).

!FILENAMES (continued)

/TRAVERSE=directory–type

(CLI32 only.) Specifies directory types to traverse (go through, or search) while executing this command. Table 2–8 contains valid values of directory type. You can use this switch to include specific directory types, such as /TRAVERSE=CPD, and to exclude directory types, such as /TRAVERSE=\CPD. Numbers from Table 2–8 are also valid values of directory type, such as /TRAVERSE=10–11. Without this switch, a command such as

```
QPRINT/TYPE=\CPD/SORT  [FILENAMES #:PROJ.–]
```

will apply to *all* directories even though /TYPE=\CPD is in the command. With this switch, commands such as the following give expected results.

```
QPRINT/SORT  [!FILENAMES/TRAVERSE=\CPD #:PROJ.–]
```

/TYPE=typecode

(CLI32 only.) Specifies one or more types of files to process. Valid values of typecode are in Table 2–8.

!FILENAMES Example 1

```
) WRITE [!FILENAMES] ⤶
...
) WRITE [!FILENAMES  #]  ⤶
...
```

The first command displays the name of each file in the working directory; the second displays the names of all files in and beneath the working directory.

!FILENAMES Example 2

```
) TYPE [!FILENAMES  TEST+.BU]  ⤶
...
```

This command displays the contents of all files in the working directory whose names begin with TEST and end with .BU.

!FILENAMES Example 3

```
) WRITE [!FILENAMES/TYPE=TXT/SORT]  ⤶
=M2N.CLI   =SET.CLI
```

This command displays the names of all text files. With CLI32, you can include the /SORT switch to display filenames in alphabetic sequence.

!FILENAMES Example 4

```
) WRITE [!FILENAMES/NOEQUAL/SORT=TYPE/SORT=–NAME] ↵
```
MY_DIR HELP_DIR 220_DIR HELP_LINK SET.CLI M2N.CLI CHECK.CLI

This command uses the /SORT= switch to display the names in ascending sequence by file type and descending sequence by file name. The first three are type DIR, the next an LNK, the next two are TXT, and the last a UDF.

!FILENAMES Example 5

In a macro:

```
[!equal [!filenames report+],]
            write There are no filenames beginning with REPORT in [!directory].
[!else]
            write Filenames beginning with report are [!filenames report+]
[!end]
```

This macro examines the working directory for filenames that begin with the characters REPORT.

A different approach, using the CLI32 switch /EXISTS, produces the same result:

```
string [!filenames/exists report+]
[!equal [!string] No]
            write There are no filenames beginning with REPORT in [!directory].
[!else]
            write Filenames beginning with report are [!filenames report+]
[!end]
```

FILESTATUS *Command*

Displays status information for specified files.

Format

FILESTATUS *[pathname ...]*

Displays information about files. If you omit arguments, the command displays information about all files in the working directory. If you include arguments, it displays information on matching pathnames only. You can use filename template characters in the pathname argument(s).

With switches, you can alphabetically sort filenames, select by date and time created or modified, display different kinds of file information, or change the file display format. If a switch does not apply to a file type (for example, the /LENGTH switch applied to link file), the CLI fills the column in which the information normally appears. CLI16 fills the column with dashes (------), CLI32 with spaces.

NOTE: FILESTATUS does not resolve link files. That is, it displays information on the link file, not on the resolution file. You can force FILESTATUS to display the resolution pathname (but no other resolution information) by using the /LINKNAME switch.

If you use a link name in a pathname, FILESTATUS neither resolves the link nor displays the resolution pathname. For example, assume file :UDD is a link to :UDD1. If you type the command FILESTATUS :UDD:+, the CLI will return no information, even though :UDD1 (the resolution file) is full of user directories. If you suspect that a filename you are using in a pathname is a link filename, verify the resolution pathname by typing a command of the form PATHNAME filename. Then use the DIRECTORY command to go to the resolution file directory and issue FILESTATUS again.

- Accepts templates.
- No argument switches.
- Requirement: *Standard*.
- See also: DIRECTORY, !FILENAMES.

Why Use It?

Use this command to list the contents of the working directory. For example, by applying one or more switches, you can limit the display to selected types of files or filenames that include certain characters.

You can also use this command to see whether or not a file exists.

FILESTATUS Command Switches

The section "Universal CLI Switches," earlier in this chapter, describes switches /1, /2, /L, /L=pathname, and /Q, which you can use with all commands. That section also explains the CLI32 switches /STR= and /ESTR=.

/ACCESSES

(CLI32 with AOS/VS II only.) Displays the number of disk accesses since files were created. (The system does not count accesses to directory, link, IPC, and device files in :PER — it displays dashes for these.) Generally, treat the accesses number as an approximate measure of file use. Details follow.

Each disk read and write request counts as an access; access to larger files may require disk access(es) for both index and data. If a file's data is already in memory (having recently been read from disk), access to this data does not change the number of accesses reported. For example, TYPE MYFILE ↵ immediately followed by TYPE MYFILE ↵ increases MYFILE's number of accesses by 1, not 2.

The access count returns to 0 if the system creates a new file to execute a command (as with LOAD_II, MOVE, or COPY, or the Link utility). Most text editor programs create a new file for the text after you edit the original file, therefore the number of accesses reported for text–edited files does not show the number of times the original file was accessed.

/AFTER/TLA=date–and/or–time
/AFTER/TCR=date–and/or–time
/AFTER/TLM=date–and/or–time

Selects files last accessed (/TLA=), created (/TCR=), or last modified (/TLM=) on or after the specified date and time (dd–mon–yy:hh:mm:ss), date (dd–mon–yy), or time (hh:mm:ss). /TCR takes a date–time value with CLI32 only. Seconds and minutes are optional. You can use /BEFORE with /AFTER to specify a span of time.

/ASSORTMENT

Displays an assortment of information — the file type, date and time of creation, and length in bytes. For link files, this switch displays the file type LNK and resolution pathname.

/AUTOSIZE	(CLI32 only.) Automatically adjusts the number of characters in the filename field to let the longest filename fit on one line. You must also use the /SORT switch. (For more on the number of characters per filename, see the /CPF switch.)
/BEFORE/TLA=date–and/or–time /BEFORE/TCR=date–and/or–time /BEFORE/TLM=date–and/or–time	Selects files last accessed (/TLA=), created (/TCR=), or last modified (/TLM=) on or before the specified date and time (dd–mon–yy:hh:mm:ss), date (dd–mon–yy), or time (hh:mm:ss). /TCR takes a date–time value with CLI32 only. Seconds and minutes are optional. You can use /AFTER with /BEFORE to specify a span of time.
/BLOCK	(CLI32 only.) Displays the number of disk blocks consumed by each file. A disk block has 512 bytes (characters).
/COUNT	(CLI32 only.) Counts the number of files listed.
/CPF=n	(CLI32 only.) Sets the number of characters per filename for display. The default is 16 characters per name. You can use this to prevent the CLI from wrapping to the next line when it displays a filename longer than 16 characters. Or if filenames displayed will have fewer than 16 characters, use it to add information (from other switches) on one line. See also /AUTOSIZE. An example of a wrapped filename display is *MY_LONG_MACRONAME.CLI* *TXT 2–Jan–92 13:02:49 4432*
/CPL=n	Sets the number of characters per line for FILESTATUS output. The default is 72. See also the /AUTOSIZE switch.
/DCR	Displays the file creation date.
/DLA	Displays the date the file was last accessed.
/DLM	Displays the date the file was last modified.

/ELEMENTSIZE	Displays the number of disk blocks in a file element for this file. A file element is the smallest unit by which a disk file can grow. (A disk block is 512 bytes.) CLI32 under AOS/VS II displays the primary and secondary element sizes and the number of primary elements in the form p:s:i, where p indicates the primary element size, s the secondary element size, and i the number of primary elements. Default numbers are set when the LDU is created; the original defaults are 4:4:1 (four blocks for the primary element size, four blocks for the secondary element size, and one primary element). The display with CLI32 and AOS/VS II for FILESTATUS/ELEMENTSIZE is *MYFILE 4:4:1*
/HASHFRAMESIZE	For AOS/VS, displays the directory's hash–frame size (whose default value is 7); see the CREATE command. For AOS/VS II, has no effect (the system ignores this switch).
/INDEX	With CLI32, displays the current and maximum number of index levels and the index element size in the form current : maximum : index–elementsize (for example, 0:3:1) With CLI16, displays the current and maximum number of index levels in the form current/maximum; for example, 0/3. The index–elementsize figure is accurate with AOS/VS II only. With CLI32 and AOS/VS, the index–elementsize displays as 0, although it is actually 1.
/INDEX	Displays the current and maximum number of index levels.
/LENGTH	Displays the length in bytes.
/LINKNAME	Displays link file resolution pathname. (By default, the CLI displays information about the link file itself. It does not display information about the resolution file.)
/NHEADER	Does not print directory headers. Also, this switch lists files with their pathnames relative to the working directory.

/NOEQUAL	(CLI32 only.) Does not print an equal symbol preceding each pathname to indicate that the pathname begins from the working directory. Useful only in conjunction with the /NHEADER switch.
/OPENCOUNT	(CLI32 only.) Displays the number of ?OPEN system calls currently effective on the file(s). Unless a file has been opened exclusively (as is routinely done by text editors), it can be opened multiple times, by the same program or different programs. For example, the CLI lets you type a file that another user is printing. You can type FILES/OPENCOUNT :CLI**.PR ⤶ to determine the number of CLI users.
/PACKET	Displays the entire contents of the packet returned by the ?FSTAT system call. (See *AOS/VS, AOS/VS II, and AOS/RT32 System Call Dictionary, ?A through ?Q.*) Most other packet information is available through other switches.
/PERMANENCE	Displays PERM if a file or directory has its permanence on. Nothing is displayed if the file's permanence is off.
/RECORD	Displays the file's record format, either as an integer (fixed–length) or as one of these mnemonics: DYN (dynamic) — a record length is specified for each read and write. VAR (variable) — each record starts with a header containing the size. D–S (data–sensitive) — records are terminated by specific characters embedded in the text.

FILESTATUS (continued)

/SORT	Sorts the file names alphabetically.
/SORT=	(CLI32 only.) Sorts display according to the key given:

BLOCK	— Number of disk blocks currently used
DCR	— Date of file creation
LENGTH	— File size (bytes)
NAME	— Filenames sorted alphabetically
OPENCOUNT	— ?OPEN count
PERMANENCE	— Permanence on file
TCR	— Date and Time of file creation
TLA	— Date and Time of last file access
TLM	— Date and Time of last file modification
TYPE	— File type

Use a minus (−) prefix to the key to sort in reverse order; and either a plus (+) prefix or no prefix to sort in ascending order. The /SORT= switch may occur up to 12 times on the command line, provided that no sort key repeats. The order in which the sort keys appear on the line indicates the order of precedence used in sorting the display. The first key is the primary sort key, the second is the secondary sort key, and so on. For example,

```
) WRITE [!FILENAMES/SORT=DCR/SORT=-LENGTH] ↵
```

displays file names in ascending order of creation date (earliest first) and, within groups having the same creation date, in reverse order of size (largest first).

/TCR*[=date:time]*	(CLI32 only.) Displays the file's creation date and time. Or with the date:time argument and /AFTER or /BEFORE, it displays the filenames created on/after or before date:time.
/TLA*[=date:time]*	Displays the date and time the file was last accessed. Or with the date:time argument and /AFTER or /BEFORE, it displays the names of files accessed on/after or before date:time.
/TLM*[=date:time]*	Displays the date and time the file was last modified. Or with the date:time argument and /AFTER or /BEFORE, it displays the names of files last modified on/after or before date:time.

/TRAVERSE=directory–type	(CLI32 only.) Specifies directory types to traverse (go through) while executing this command. Common types are DIR (directory), CPD (control point directory), LDU (logical disk unit), and LNK (link). For all types, Table 2–8 contains valid values of directory–type. You can use this switch to include directory types, such as /TRAVERSE=CPD, and to exclude directory types, such as /TRAVERSE=\CPD.
	
	Without this switch, a command such as
	
	`FILESTATUS/TYPE=\CPD #:PROJ.–,RE`
	
	will apply to *all* directories even though /TYPE=\CPD is in the command.
	
	With this switch, commands such as the following give expected results.
	
	`FILESTATUS/TRAVERSE=\CPD #:PROJ.–,RE`

/TYPE	Displays the file's type, either as a three-character mnemonic, or as a decimal number (0–255) if a mnemonic doesn't apply.
/TYPE=typecode	Selects files of the specified type, where the typecode is a 3-letter mnemonic or integer that represents a file type. Common types are DIR (directory), CPD (control point directory), LDU (logical disk unit), and LNK (link). For all types, see Table 2–8. To specify a range of file types, use the format /TYPE=typecode–typecode To exclude a file type or range of file types, precede the type code with a backslash, as in /TYPE=\typecode or /TYPE=\typecode–typecode You can use more than one /TYPE switch in a command.
/UDA	Displays the characters in the User Data Area if the file has a UDA.
/XPACKET	(AOS/VS II with CLI32 only.) Displays the entire contents of the extended packet as returned by the ?XFSTAT system call. This switch returns information not supplied by /PACKET, which displays ?FSTAT information. (See *AOS/VS, AOS/VS II, and AOS/RT32 System Call Dictionary, ?R through ?Z.*) Most other packet information is available through other switches.

FILESTATUS Example 1

```
) FILESTATUS/ASSORTMENT/SORT ⤶
```

```
Directory :UDD1:ANDY:CLI:COMMANDS

   ACL              TXT   22-Jul-91  11:30:38      3949
   ACL_COM          UDF   24-Jul-91  07:19:52     50569
   CON_DUZ          UDF   24-Jul-91  12:37:50     50011
   .

   .
   RDO_SWA          TXT   22-Jul-91  11:15:54     48621
   SAMPLE           TXT   24-Jul-91   9:22:48     25790
   SYS_XEQ          TXT   18-Jul-91  13:02:10     44776
```

This command displays a list of files in the working directory. Filenames appear in alphabetical order together with the file type, date and time of creation, and length in bytes of each file.

FILESTATUS Example 2

```
) FILESTATUS/TYPE=DIR ⤶
```

```
Directory :UDD1:ANDY

 INTRO    MACROS  CLI        MTV
```

This command displays the name of each directory (file type DIR) contained in the working directory. It does not display names of directories of type CPD and LDU.

FILESTATUS Example 3

```
) FILESTATUS/AFTER/TLM=1-JAN-92/SORT +\+.ED ⤶
```

```
Directory :UDD1:ANDY:CLI

   APDXE        CHAP1        CHAP2        CHAP3
   CHAP4        CHAP6        CLARK        CLI.DOC
   COMMANDS     COMSAP       CYNTHIA      ERIN
   FSPEC        FSPEC        JASON         LEO
   UPDATE.CLI
```

This command shows an alphabetical list of files in the working directory that were modified any time after 1 January 1992 — excluding all filenames ending in .ED (SED text editor status files).

!FILESTATUS Example 4

```
) FILESTATUS/ASSORTMENT/SORT=TYPE/SORT=–NAME ⏎

    CLI.PR            PRV   28–Oct–91  12:02:54     534528
    CLI.ST            STF   28–Oct–91  12:02:54      63488
    CS.CLI            TXT   16–Jul–91  14:37:21        544
    CLI32.PARAMS.OB   UDF   25–Oct–91   9:59:32     170620
    CLI32.PAR         UDF   25–Oct–91   9:57:16    1122304
    CLI.DS            UDF   28–Oct–91  12:02:29     978944
    CLI.DL            UDF   28–Oct–91  12:02:04      10240
```

This command displays an assortment of information for files sorted in ascending order of file type and descending order of file name.

HISTORY

Command

Displays and retrieves commands (CLI32 only).

Format

HISTORY *[command_number]*

This command saves command lines entered from your terminal. The command lines go to an area of memory known as the history buffer. Each command line is numbered in the buffer. You can display or retrieve a command line by pressing the cursor control uparrow or downarrow key, or by referring to the command's line number.

By default the history buffer holds 25 command lines, but you can change this number by using the HISTORY /SAVE switch. A command line can contain one or more commands to the CLI. So, all the following commands require one numbered line in the history buffer.

```
) DIRECTORY; DATE; TIME; WHO )
```

A command line can also contain part of a command. For example, you can give the DIRECTORY command as

```
) directo& )
&)ry )
```

and it will occupy two lines in the history buffer.

The command lines go into the buffer sequentially with the most recent commands having the higher numbers. For example, suppose you have just logged on and given the following five commands (where the CLI responses do not appear).

```
DATE
TIME
DIRECTORY MYDIR
QPRINT/COPIES=3 MY_FILE_1 MY_FILE_5
QDISPLAY/QUEUE=LPT
```

In this case the correspondence between command number and command is

Command Number	Command
1	*DATE*
2	*TIME*
3	*DIRECTORY MYDIR*
4	*QPRINT/COPIES=3 MY_FILE_1 MY_FILE_5*
5	*QDISPLAY/QUEUE=LPT*

There is room for 20 more commands, numbered from 6 through 25, in the buffer. If you typed these 20 commands the buffer would be full. If you then typed two more they would displace the earliest ones; the first five command lines and their reference numbers would be as follows.

3	*DIRECTORY MYDIR*
4	*QPRINT/COPIES=3 MY_FILE_1 MY_FILE_5*
5	*QDISPLAY/QUEUE=LPT*
6	*First of the next 20.*
7	*Second of the next 20.*

The /RENUMBER switch, described later, renumbers the commands in the buffer so that first one is number 1.

Other properties of the history buffer and the HISTORY command are

- Command numbers in the history buffer start at 1 and increase by 1 with a maximum value of 4,294,967,295.
- The CLI does not place null lines in the history buffer. A null line is a line that you create by pressing just one key — a delimiter, such as NEW LINE. The CLI does save lines consisting only of white space, such as spaces and tabs.
- If you enter a command and press the uparrow (↑) or downarrow (↓) key before pressing a delimiter key, the CLI erases the text. It also displays the appropriate command from the history buffer. For example, suppose the history buffer contains the following commands.

  ```
  DATE
  TIME
  DIRECTORY MYDIR
  FILESTATUS/ASSORTMENT MISCELLANEOUS+
  ```

  If you give the command and terminator

  ```
  ) WRITE ASDFJKLQWERT ↑
  ```

  (where ↑ represents the uparrow key, not the character generated by pressing Shift–6 on the keyboard), the CLI responds by discarding the WRITE command and displaying the last command in the buffer:

  FILESTATUS/ASSORTMENT MISCELLANEOUS+

  If you press NEW LINE the CLI executes this FILESTATUS command. On the other hand, if instead you gave the command and terminator

  ```
  ) WRITE ASDFJKLQWERT ↓
  ```

  (where ↓ represents the downarrow key), the CLI responds by discarding the WRITE command and displaying the first command in the buffer:

  DATE

  If you press NEW LINE, the CLI executes this DATE command.

In general, the CLI treats the list of commands as a circular buffer so that the uparrow key moves upward through the history buffer and the downarrow key moves downward through the buffer. If, in this example, you had terminated the command

```
) WRITE ASDFJKLQWERT↑↑
```

with two consecutive uparrows, the CLI would have displayed the FILESTATUS command and then the DIRECTORY command.

You need not type any command before pressing the uparrow or downarrow key. If, in this example, you had responded to the CLI prompt with two consecutive downarrows, the CLI would have displayed the DATE command and then the TIME command.

You can set the number of commands to save at any time with the /SAVE=n switch, described later. If the new number is smaller than the old number, the CLI truncates the history buffer. This means that it removes the oldest commands from the history buffer.

The PREFIX command has a /HISTORY switch that accepts the value On or Off. The command

```
) PREFIX/HISTORY=ON⤶
```

displays an exclamation point (!) and the number of the current command in the CLI prefix. For example, if the prefix is "Hello" and you decide to give the DIRECTORY command as the current command, you might see

14!Hello DIRECTORY

just before you press the NEW LINE key. The default value of this /HISTORY= switch is Off; so, you don't see a number and an exclamation point (!) just before the prefix unless you use the /HISTORY switch.

- No templates.
- No arguments.
- Requirement: *Standard.*
- See also: PREFIX.

Why Use It?

Use the HISTORY command to repeat CLI commands you made earlier and to keep track of commands. As the previous explanation of the uparrow and downarrow keys shows, you can gain access to and execute CLI commands that are in the history buffer. HISTORY with the /WRITE switch copies commands from the history buffer into a macro file for later execution.

HISTORY Command Switches

The section "Universal CLI Switches," earlier in this chapter, describes switches /1, /2, /L, /L=pathname, /Q, /STR=, and /ESTR= which you can use with all CLI32 commands.

/COMPRESS=x	Enables or disables compression of the commands in the history buffer. For x, use ON or OFF. The default is ON. With compression enabled, a command line that is *exactly* the same as the last line in the history buffer is not written into the buffer. This mode does not remove any commands already in the buffer. If you specify /COMPRESS=OFF, all command lines are written to the buffer.
/KILL	Clears the history buffer by erasing all command lines in the buffer and resetting the next command number to 1.
/READ=filename	Places the contents of the specified file into the history buffer,. just as if you typed them. For example, suppose the size of the history buffer is six lines and file XX.CLI contains eight lines. If you give the command `) HISTORY/READ=XX.CLI↵` then the CLI tries to place the entire file in the history buffer. There is not room, so the oldest command lines (first lines read) from the file are overwritten and the newest (that is, last read) six lines become the history buffer. If the size of the history buffer is ten lines, then the two newest lines of the history buffer move to its top and the lines from the file move into the bottom.
/RENUMBER	Resets the numbers of the command lines so that the first command line is number 1.
/SAVE*[=n]*	Without n, displays the number of commands stored in the history buffer. With n, sets the number of commands that can be stored in the history buffer. The default value of n is 25.

/WRITE=filename *[command_number ...]*

Writes the specified commands, without the corresponding values of command_number, into filename. Later, filename will function just like a macro file. For example,

```
) HISTORY/WRITE=MYMACRO.CLI  3, 8, 12, 5↵
```

The CLI displays an error message if command_number is outside the range of valid command numbers. This would happen if the size of the history buffer was 20, the first command number was 31, and you specified command 55.

If you omit command_number, the CLI writes the entire history buffer into (including the HISTORY/WRITE command) filename. The file filename cannot already exist.

/LDUNAME=filename	Specifies the filename of the LDU you want to initialize. You need this switch only if a disk unit you specify contains more than one LDU. You must also precede the disk unit names with the LDU unique ID and a slash. For example, assume that disk unit DPJ11 holds two LDUs, named DATA1 (unique ID DATA1.IMAGE1) and DATA2 (unique ID DATA2.IMAGE2). To initialize DATA1, you would type *Su*) INITIALIZE/LDUNAME=DATA1 DATA1.IMAGE1/@DPJ11 ⤸
/NOHARDWARE	Forces the system to use software mirroring (the operating system, not the disk controller, does the mirroring I/O). By default, if you omit this switch, the system tries to use hardware mirroring; then, if hardware mirroring is not possible, it tries to use software mirroring. Use this switch if you want software mirroring even if hardware mirroring is possible.
/NOMIRROR	Initializes one image of a mirrored LDU. Use this switch when the images are out of synchronization; then use the MIRROR command to synchronize the other image(s).
/S	Places the LDU name in the CLI String instead of displaying it on the terminal. (See the STRING command.)
/TRESPASS	Initializes the LDU even though it is marked as owned by another system. (The disk was last initialized by another system.) This switch is designed for systems that use multiported disks. Use the switch only if your system needs access to the multiported disk information and you're sure the other system has failed. If the other system is still running and you use this switch, disk information will be corrupted.

INITIALIZE (AOS/VS II) Example 1

The following commands initialize a single-disk LDU, filename UDD1, in disk unit DPJ1, in the root directory.

```
) SUPERUSER ON ⤸
Su) INITIALIZE/DIR=: @DPJ1 ⤸
UDD1                              (System displays LDU filename.)
```

INITIALIZE (AOS/VS II) Example 2

The next example initializes as in the example earlier, and also starts mirroring the LDU image in DPJ1 to the one in DPJ2.

```
) SUPERUSER ON ↵
Su) INITIALIZE/DIR=: @DPJ1!@DPJ2 ↵      (The system displays the LDU filename
                                         and a confirmation message, as follows.)
UDD1
From system:
LDU 'UDD', image 'UDD.IMAGE2' is hardware mirrored -- synchronized
```

INITIALIZE (AOS/VS II) Example 3

Suppose there is a two-disk LDU named DATABASE in disk units DPJ1 and DPJ2. This example shows the message that returns when a person specifies only one disk of this LDU; then it shows the correct command.

```
) SUPERUSER ON ↵
Su) INITIALIZE @DPJ1 ↵                        (Try one unit.)
Error: Incomplete logical disk unit (LDU)     (Error.)

Su) INITIALIZE @DPJ1 @DPJ2 ↵                  (Try two units.)
DATABASE                                      (Success.)
```

INITIALIZE (AOS/VS II) Example 4

Suppose the disk in unit DPJ22 has two LDUs on it, SAM (with unique ID SAM.IMAGE1) and CAROL (with unique ID CAROL.IMAGE1). The following sequence of commands initializes both LDUs. The system displays the LDU filename after each command succeeds.

```
Su) INITIALIZE/LDUNAME=SAM SAM.IMAGE1/@DPJ22 ↵
SAM

Su) INITIALIZE/LDUNAME=CAROL CAROL.IMAGE1/@DPJ22 ↵
CAROL
```

You can check the contents of a tape label using the TYPE command or display utility; for example,

```
) TYPE @MTB0)
```

The volid follows the word VOL1; the labeled tape filename follows the word HDR1.

If you will be mounting your own tapes and writing your own labels and dumps, you must have access to the system console and act as your own system operator (using CONTROL EXEC commands, for instance).

NOTE: Dump density and label density must match; if they do not, you will get the message that there is a density mismatch. So, plan ahead; if you want to dump at 6250 BPI, prepare labeled tapes by setting the density switch to 6250, or use the LABEL program's /DENSITY switch.

CAUTION: Labeling a tape makes reading the previous contents of the tape impossible.

- Does not accept templates.
- Accepts utility switches (described later).
- Requirement: *Standard* (Execute access to the program file in :UTIL).
- See also: MOUNT with /VOLID switch, DUMP or DUMP_II, LOAD or LOAD_II, OPERATOR.

Why Use It?

Use this utility to write a label onto a magnetic tape or diskette. You may want to do this if you have a collection of data that will span several volumes. In this situation, labels help to ensure that data is read or written in the correct order and that none of the data is missing. If you use unlabeled media, your data must fit on one physical volume.

Labeled tapes and diskettes also allow you to refer to a file by a volume ID (volid) and filename rather than by file number.

Generally, use LABEL to prepare labeled tapes rather than diskettes. For diskette labeling, the CLI's labeling feature is easier to use (as described under the OPERATOR command).

For more information on labeled media, see Chapter 6 of this manual. You can find more information about labeled tapes in the manual *Managing AOS/VS and AOS/VS II.*

LABEL Switches

You must use a switch's full name instead of an abbreviation.

/ACCESS=ascii–char	Writes an ASCII character into the access control byte (offset ?LBAC) in the parameter packet for system call ?LABEL. This switch is meaningful only for tapes that will be read by operating systems that use ANSI standards for labeled tapes. To specify full access, use the ASCII space character.
/DENSITY=value	Specifies the density at which data will be dumped to this tape, overriding the system default. This switch is for use with variable density tape drives. The value can be one of the following:

800	(800 bytes per inch)
1600	(1600 bytes per inch)
6250	(6250 bytes per inch)
ADM	(Automatic Density Matching)
LOW	
MEDIUM	(reverts to LOW on a dual-density unit)
HIGH	

If you intend to use a tape on another unit, be careful about choosing LOW, MEDIUM, or HIGH: "low" or "high" on one unit may not be compatible with the values on another unit, which would prevent reading from the tape.

You must write data to the tape at the same density used to write the label. By default, the system writes the label at the default density, which is the value selected for the tape unit during VSGEN.

/I	Creates an IBM label. Use this switch only if the volume will be used on a system that can read IBM-format labels. The default label format is ANSI compatible: suitable for DGC and many other non-IBM systems. Make sure to use the /IBM switch with the DUMP_II and LOAD_II commands. *CAUTION:* *Be sure that the destination system requires and supports IBM format before you use this switch. You cannot use labeled tape access after changing the label format.*
/LEV=n	Creates a label at ANSI level n where n is 1, 2, 3, or 4. The default level is 3.
/MAXCAP	Specifies streaming mode on Model 6352 cartridge tapes only. The result is faster I/O and greater capacity. If you're using a Model 6352 unit and want the unit to stream later when it writes information to the tape, you must use this switch. Otherwise, the unit is limited to slower start/stop mode.
/OWNER=string	Writes the specified string in the label's owner field. For a DG or ANSI label, the owner field can be 14 characters; an IBM label restricts the owner field to 10 characters.
/PERFECT	Diskette only. Tells the LABEL program to check the entire diskette for bad blocks; LABEL rejects the diskette if it finds any bad blocks. By default, LABEL checks only the label area and aborts if it finds any bad blocks there.
/REMAP	Diskette only. Tells the LABEL program to check the entire diskette for bad blocks; LABEL will remap up to two bad blocks but reject the diskette if it finds more than two bad blocks. By default, LABEL checks only the label area and aborts if it finds any bad blocks there.
/S	Scratches the volume (by writing beginning and end-of-file labels for a null first file). This process effectively deletes all files on the volume; you must include the volid to ensure that the utility is scratching the correct volume.

LABEL (continued)

/UVL=string	Writes a user volume label of the specified string into the label. The string can be as many as 76 characters. You can specify up to nine user volume labels using multiple /UVL switches. By default, the UVL fields are left blank. User volume labels are not supported in IBM format tapes. So, if the XEQ LABEL command contains both the /I and /UVL switches, LABEL ignores the /UVL switches.

LABEL Example 1

```
) XEQ LABEL/OWNER=LEE  @MTB0  VOL1↵
```

(Remove the newly labeled tape and mount the next tape.)

```
) XEQ LABEL/OWNER=LEE  @MTB0  VOL2↵
```

The first command writes a label on the tape mounted on MTB0 and assigns it a volid of VOL1. After the first tape is dismounted and replaced with the next, the second command labels the new tape and assigns it a volid of VOL2.

LABEL Example 2

The following command sequence labels two tape volumes. Then, it requests a user tape mount from the EXEC utility and dumps files to the tapes.

```
) SUPERUSER ON↵
Su) MOUNT/VOLID=VOL1/VOLID=VOL2/EXTEND  XTAPE  Please↵
```

(Mount the first tape. At the system console, type a command of the form
CX MOUNTED unitname)

```
Su) DIR  :↵
Su) DUMP_II/BUFFERSIZE=16384/V/L=:UDD:[!USERNAME]:SYSTEM_BACKUP &↵
&) :UDD:[!USERNAME]:XTAPE:FILESET1↵
```

(I/O proceeds through VOL1, and after the EXEC prompt and tape change, VOL2.)

```
Su) DISMOUNT  XTAPE  Backup is done.↵
```

LABEL Example 2 (continued)

This example shows a system backup, with a multiple volume labeled tape dump. The MOUNT command asks the system operator (person at the system console) to mount tape volume V1 (first of a sequence of two volumes). The DIRECTORY command makes the root directory the working directory. The DUMP_II command then dumps all files in the system, maintaining directory structure, to tape file FILESET1 using the name XTAPE (linkname XTAPE is created in the initial user directory). The dump can use up to three tape volumes. The dump listing goes to file SYSTEM_BACKUP in the person's initial user directory. After the dump is done, the person types DISMOUNT, prompting the operator to dismount the last tape.

LABEL Example 3

This example shows labeled tape access with a Model 6352 unit.

Mount tape on tape unit MTJ0.

```
) XEQ LABEL/MAXCAP @MTJ0 V1↵          (Label a tape V1, with /MAXCAP.)

) MOUNT/VOL=V1 XX PLEASE↵             (Request mount of the tape.)
```

Mount V1 on unit. On the system console, type

```
) CONTROL @EXEC MOUNTED @MTJ0↵
```

From a user console, start writing to the labeled tape:

```
) DELETE/2=IGNORE SAM.LS↵
) XEQ DUMP_II/V/L=SAM.LS/MAXCAP :UDD:SAM:XX:LFILE SAM:#↵
```

Dump to the tape volume, using tape filename LFILE (the link pathname is :UDD:SAM:XX), with listing to file SAM.LS. Dump directory SAM and all its files.

```
) DISMOUNT XX PLEASE↵                 (Request tape dismount.)
```

LABEL Example 4

This example shows creation of a labeled tape to be read on an IBM system.

Mount tape on tape unit MTB0.

```
) XEQ LABEL/I @MTB0 BBLUE↵                (Label tape as volume BBLUE,
                                           IBM format.)
) MOUNT/VOL=BBLUE/IBM TTAPE Please↵       (Request mount of the IBM tape.)
```

Volume BBLUE remains mounted on unit MTB0. On the system console, type

```
) CONTROL @EXEC MOUNTED @MTB0↵
```

On a user terminal, change working directory and dump to the labeled tape:

```
) DIR :UDD:TRANSIT↵                       (Set directory for dump.)

) DELETE/2=IGNORE LIST↵
) XEQ DUMP_II/V/L=LIST/IBM TTAPE:TRANSIT↵
```

Dump to the tape, using tape filename TRANSIT, with listing to freshly-created file LIST. There is no template, so it dumps all files in the working directory.

```
) DISMOUNT TTAPE↵                         (Request tape dismount.)
```

LOAD *Command*

Loads the specified files from a dump file into the working directory (CLI16 only).

Format

LOAD dumpfile *[source–pathname] [...]*

LOAD restores files that were dumped to tape or disk via the DUMP command or DUMP_II utility. The dump file can be

- a file on a magnetic tape, such as @MTB0:0;
- a tape linkname, such as MYTAPE (after you use the MOUNT command);
- a disk file pathname, such as DUMPDIR:DUMPFILE;
- a devicename, such as @DPJ10, or the labeled diskette file, @LFD.

NOTES: Do your best to ensure that the working directory is appropriate for the load. This is particularly important if the dump file contains a directory structure, since the utility will try to duplicate the dump–file structure within the working directory. For example, if directory UDD:ALEX:REPORTS was dumped from the root directory, and the working directory when you load this dump file is :UDD:ALEX (instead of the root), the load will end with REPORTS in directory :UDD:ALEX:UDD:ALEX:REPORTS.

If you have doubts, use the /N switch to check directory structure at the beginning of the dump file; then change the working directory as needed.

The LOAD command does not try to match buffer sizes. If a nonstandard buffer size (other than 2048) was used for the dump, LOAD will display the error message *Indecipherable dump format.* If you see this error message, retry the command with /BUFFERSIZE=8192 and, if that fails, with /BUFFERSIZE=32768 (these are the most common maximum buffer sizes). This problem is particularly likely with a dump file created by DUMP_II, since that utility tries to use as large a buffer size as possible. The problem does not occur with LOAD_II, which tries to match the buffer size used for the dump.

If you omit arguments and date switches, the CLI tries to load the whole dump file, with its directory structure, into the working directory. You can specify pathnames (including templates) for files in the dump file; for example

```
) LOAD/V @MTB0:0 UDD:CHRIS:#⤶
```

To display pathnames in the dump file without loading them, use the /N switch. To load the files in the dump file without maintaining the dumped directory structure, use the /FLAT switch.

To load specific files, you must use a template that matches the pathname in the dump file. Thus, if you want to load files matching DIR1:MYF+ that were dumped from directory JR, you must specify DIR1:MYF+. If the files were dumped from the root directory, you must specify UDD:JR:DIR1:MYF+.

- Accepts templates for source–pathname.
- No argument switches.
- Requirement: *Standard* (Execute and Write or Append access to the working directory).
- See also: LOAD_II, DUMP, DUMP_II, MOUNT, OPERATOR.

Why Use It?

Use the LOAD command to retrieve files from any tape, disk, or diskettes to which files were dumped with the DUMP or DUMP_II command.

The LOAD_II utility program (next) is faster than LOAD, but it does not support labeled diskettes. To load from labeled diskettes, you must use the LOAD command. The LOAD command is available from CLI16 only. (With CLI32, a LOAD.CLI macro executes the LOAD_II program.)

The LOAD command cannot load the second or subsequent volume or an unlabeled tape set created with the DUMP_II utility. If you want to load from all volumes of such a tape set, use LOAD_II. To load files that were dumped on a UNIX system, see CPIO_VS or TAR_VS.

Command Switches

The section "Universal CLI Switches," earlier in this chapter, describes switches /1, /2, /L, /L=pathname, and /Q, which you can use with all commands.

/AFTER/TLA=date–and/or–time /AFTER/TCR=date–and/or–time /AFTER/TLM=date–and/or–time	Selects files last accessed (/TLA=), created (/TCR=), or last modified (/TLM=) on or after the specified date and time (dd–mon–yy:hh:mm:ss), date (dd–mon–yy), or time (hh:mm:ss). Seconds and minutes are optional. You can use /BEFORE with /AFTER to specify a span of time.
/BEFORE/TLA=date–and/or–time /BEFORE/TCR=date–and/or–time /BEFORE/TLM=date–and/or–time	Selects files last accessed (/TLA=), created (/TCR=), or last modified (/TLM=) on or before the specified date and time (dd–mon–yy:hh:mm:ss), date (dd–mon–yy), or time (hh:mm:ss). Seconds and minutes are optional. You can use /AFTER with /BEFORE to specify a span of time.

/BUFFERSIZE=n	Sets the I/O buffer size to n bytes, up to the limit set at system generation. ECLIPSE MV/3000 series systems and above allow a maximum of 32768. On all systems, 21-, 120-, 150-, 320-, and 525-Mbyte cartridge tapes allow a maximum of 16384. The buffer size used for the load must match the buffer size used for the dump. If not, the program will stop with an error message. Legal buffer sizes for dumps are multiples of 1024. You can also specify this as mK; for example, 8K means 8192 bytes. The default buffer size for DUMP and DUMP_II is 2048 (or for dumps to MTJ units, 16384).
/DELETE	Deletes any existing file with the same name (unless it is a directory file, in which case LOAD will simply load files into the directory). If you use /VERIFY, the CLI will verify deletions.
/DENSITY=	Specifies tape density. The density was established when the tape was dumped; you cannot change the density on the tape. You should omit this switch; if you must use it, use /DENSITY=ADM (automatic density matching).
//FLAT	Loads all files directly into the working directory; does not maintain the directory structure in the dump file.
/IBM	Loads from a tape with an IBM–format label.
/N	Lists the files that would otherwise be loaded, but does not actually load them.
/NACL	Does not load access control lists; the CLI assigns the current default ACL to all files that it loads.
/RECENT	Loads only those files created more recently than existing files of the same name, unless the existing file is a directory. (Comparison is based on the time created, not the time last modified.)
/SEQUENTIAL	For labeled tape that has been mounted via MOUNT/VOLID= and the EXEC command MOUNTED. Prevents the program from rewinding the tape after completing the load. This saves rewind and spool forward time if you want to load again from the same tape volume.
/SPECIFIC	Loads from the tape volume specified (you must use an implicit mount via @LMT:volid:tape–filename). Use this switch to load from a logical file within a file set without going through all preceding volumes. This switch is further described in Chapter 6.

/TYPE=typecode	Loads files of the specified type, where the type code is either a 3-letter mnemonic or a decimal integer that represents a specific file type. (See Table 2–8.) To specify a range of file types, use the format /TYPE=typecode–typecode; to exclude a file type or range of file types, precede the type code with a backslash (/TYPE=\typecode or /TYPE=\typecode–typecode). You can use more than one /TYPE= switch in a command.
/V	Displays the name of each loaded file on @OUTPUT. When used with /DELETE or /RECENT, this switch verifies the deleted files. The filenames go to your terminal or file specified with the /L= switch.

LOAD Example 1

```
) LOAD/V/NACL  @MTD0:0⤶
```

This command loads the contents of tape file 0 (which is mounted on drive MTD0) into the working directory. The switches cause the CLI to display the name of each file as it is loaded, to ignore any ACLs on the files, and to assign the current default ACL to the files.

LOAD Example 2

```
) LOAD/V @MRCTAPE0000A0:1  +.PR⤶
```

This command loads into the working directory all the files with a .PR filename suffix that are stored in file 1 of the tape mounted on unit @MRCTAPE0000A0. Each filename is displayed as the file is loaded.

LOAD Example 3

```
) LOAD/N  @MTJ0:0⤶
... (system displays filenames in dump file without loading them) ...
```

The second command rewinds the tape. The command simulates loading the contents of tape file 0 without actually copying the contents to the working directory (because of the /N switch).

LOAD Example 4

```
) LOAD/V/BUFFER=8192/AFTER/TLM=22-MAY-90:12/TYPE=\LNK &⤶
&)    @MTC0:2  MYDIR:#⤶
```

This loads from the third file of the tape on unit MTC0, which was dumped (thus must be loaded) with a buffer size of 8192 bytes. The CLI loads all files created or modified on or after noon, May 22, 1990, excluding links, from directory MYDIR.

LOAD Example 5

```
) SUPERUSER ON⤶
Su) MOUNT/VOLID=V1/VOLID=V2/VOLID=V3  MYTAPE  Ready to restore⤶
```

... (System operator mounts first tape.)

```
Su) DIR :⤶
Su) LOAD/BUFFERSIZE=16384/V/R/L=:UDD:CHRIS:FILES_RESTORED  &⤶
&Su)   :UDD:[!USERNAME]:MYTAPE:FILESET1   UDD:CHRIS:PROJECTS:#⤶
```

... (System reads from multiple volumes, loads files.)

```
Su) DISMOUNT MYTAPE  Restoration is done.⤶
```

This shows the restoration of directory :UDD:CHRIS:PROJECTS and its files from a system backup done on labeled tape. The MOUNT command asks the system operator (person at the system console) to mount tape volume V1 (first of a sequence of three volumes). The DIRECTORY command makes the root directory the working directory — needed since the system backup was done from the root directory. The LOAD_II command tells the program to search the dump file named FILESET1, via tape linkname MYTAPE (in user's initial directory), for the specified directory and load it. A listing of restored files goes to file FILES_RESTORED in CHRIS' directory. After the load, the DISMOUNT command asks the operator to dismount the tape(s).

LOAD_II *Utility*

Loads one or more previously dumped files into the working directory.

Format

LOAD_II dumpfile *[source–pathname] [...]*

The LOAD_II utility restores files that were dumped to tape or disk using the DUMP_II utility or the DUMP command. The dump file can be one of the following.

- a file on a magnetic tape, such as @MTB0:0
- a tape link name, such as MYTAPE (after you use the MOUNT command)
- a disk file, such as DUMPDIR:DUMPFILE
- a device name, such as @DPJ10 (but not the labeled diskette file, @LFD).

You can specify pathnames (including templates) for files in the dump file. If you omit source–pathname arguments and date switches, the utility tries to load the whole dump file, with its directory structure, into the working directory.

To load files in the dump file without maintaining the original dumped directory structure, use the /FLAT switch.

NOTE: Think carefully about your working directory before starting a load operation. This is particularly important if the dump file contains a directory structure, since the utility will try to duplicate the dump–file structure within the working directory. For example, if directory UDD:ALEX:REPORTS was dumped from the root directory, but the working directory when you load this dump file is :UDD:ALEX, the load will end in directory :UDD:ALEX:UDD:ALEX:REPORTS.

If you have doubts about whether your current working directory is appropriate, use the /N switch to check the directory structure of the dump file before loading any files.

To load specific files, you must use a template that matches the pathname in the dump file. Thus, if you want to load files matching DIR1:MYF+ that were dumped from directory JR, you must specify DIR1:MYF+. If the files were dumped from the root directory, you must specify UDD:JR:DIR1:MYF+.

LOAD_II and a macro to execute it are supplied in the root directory. In addition, a link to the utility and a copy of the macro to execute it are supplied in directory :UTIL.

- Accepts templates.
- Accepts utility switches (described later).
- Requirement: *Standard* (Execute access to the program file in :UTIL or the root); you also need Execute access and Write access or Append access to the working directory; or you need Superuser on.
- See also: MOUNT, DUMP_II.

Multiple-Volume Loads

LOAD_II can load from multiple–volume unlabeled tape dump files that were created by DUMP_II. (It can load from multiple–volume labeled tape files created by DUMP_II or DUMP.) When LOAD_II loads from unlabeled tape and reaches the end of a volume without reaching the end of the dump file, it prompts for remaining volumes by displaying

Mount the next volume, volume: n
Respond MOUNTED <device> or REFUSED when ready.

To continue the load, you can then mount the next tape — on the same unit or on a different unit. If you use the same unit, you can simply type

```
MOUNTED)
```

or an abbreviation, omitting a device name. If you use a different unit, you must also type the unit name. (With unlabeled tape, the utility cannot check the volume mounted, so don't confirm by typing MOUNTED until you're sure the correct tape is mounted.)

Note that when DUMP_II continues a dump on a new tape, the continuation begins at tape file 0 on the new tape; if you issue another LOAD_II command to this tape volume, you must specify tape file 1. For example, you issue three load commands (LOAD_II ... @MTD0:0, LOAD_II ... @MTD0:1, and LOAD_II ... @MTD0:2), and the third load continues to another tape; when the third load completes, you must specify tape file 1 in the next load command (*not* file 3), since the continuation started at file 0 on the new tape.

LOAD_II can recover from most hard tape errors. On a hard tape error, LOAD_II can skip the unreadable part of the tape and continue with the next readable tape block.

LOAD_II (continued)

In batch mode, LOAD_II always begins by trying to create an IPC file in your working directory. It might need this file later to communicate with the system console. You should have Write access to your working directory or have Superuser privilege turned on. If LOAD_II can't create this IPC file initially, it immediately aborts with a termination message — because Write access, needed to load the files, is required.

LOAD_II communicates with the system console in batch mode when the load requires more than one reel of unlabeled tape or to report errors. LOAD_II then sends an informative message to the system console and awaits a response. The operator normally responds with a message of the form

CONTROL :UDD:username:LOADpid MOUNTED tape–unitname

to continue the load. Or, the operator can refuse to mount another reel, thus ending the load operation abnormally.

For example, suppose you are user JEFF and execute LOAD_II in batch mode. Furthermore, suppose LOAD_II runs as PID 71 and has loaded the first reel of unlabeled tape from drive @MTB2 during a load operation. It displays the following on the system console.

```
From Pid 71:   (LOAD00071)   The tape is rewinding ...
From Pid 71:   (LOAD00071)   Mount the next volume, volume: 02
From Pid 71:   (LOAD00071)   Respond CONTROL :UDD:JEFF:LOAD00071
From Pid 71:   (LOAD00071)   Respond MOUNTED <device> or REFUSED when
ready.
```

To continue the load, you (or the operator) change tape reels and respond with

```
) CONTROL :UDD:JEFF:LOAD00071 MOUNTED @MTB2↵
```

To stop the load, you (or the operator) respond with

```
) CONTROL :UDD:JEFF:LOAD00071 REFUSED↵
```

Why Use It?

Use the LOAD_II utility quickly to restore files dumped to tape or disk with either the DUMP_II utility or the DUMP command.

The LOAD_II utility is superior to the LOAD command for restoring files because it lets you recover from hard tape errors, it lets you load from multiple unlabeled tape volumes, and because it is faster than LOAD. However, LOAD_II cannot use labeled diskettes; if you want to use labeled diskettes, you must use the DUMP command. The DUMP command is available from CLI16 only.

To load files that were dumped on a UNIX system, see CPIO_VS or TAR_VS.

LOAD_II Switches

The section "Universal CLI Switches," at the beginning of this chapter, describes the /1, /2, /L, /L=pathname, and /Q switches, which you can use with the command line that invokes this utility program. Switches /1= and /1= must occur first on the command line. Otherwise, error handling may not occur properly.

/AFTER/TLA=date–and/or–time
/AFTER/TCR=date–and/or–time
/AFTER/TLM=date–and/or–time

Selects files last accessed (/TLA=), created (/TCR=), or last modified (/TLM=) on or after the specified date and time (dd–mon–yy:hh:mm:ss), date (dd–mon–yy), or time (hh:mm:ss). Seconds and minutes are optional. You can use /BEFORE with /AFTER to specify a span of time.

/BEFORE/TLA=date–and/or–time
/BEFORE/TCR=date–and/or–time
/BEFORE/TLM=date–and/or–time

Selects files last accessed (/TLA=), created (/TCR=), or last modified (/TLM=) on or before the specified date and time (dd–mon–yy:hh:mm:ss), date (dd–mon–yy), or time (hh:mm:ss). Seconds and minutes are optional. You can use /AFTER with /BEFORE to specify a span of time.

/BLOCKCOUNT=n	Specifies the number of blocks LOAD_II will read from the dump medium in one input operation. A block equals the buffer size (if loading from tape) or 512 bytes (if loading from disk). Values range from 1 block (default) to 255. If you use 255 blocks, the program will use as many blocks as possible, and may therefore speed up loading from a disk file. When you are loading small files from tape, use this switch in conjunction with a small buffer size (such as 2048, the default) to improve performance.
/BOTH[*=pathname]* or /B[*=pathname]*	Lists information both to your terminal and file pathname. Omit pathname to use the generic @LIST file. If pathname doesn't exist, LOAD_II creates it; if it does exist, LOAD_II appends to it. To display filenames, you must also use the /VERBOSE switch, which you can abbreviate to /V.
/BUFFERSIZE=n	Sets the I/O buffer size to n bytes, up to the limit set at system generation. ECLIPSE MV/3000 series systems and above allow a maximum of 32768. On all systems, 21-, 120-, 150-, 320-, and 525-Mbyte cartridge tapes allow a maximum of 16384.
	By default, LOAD_II matches the buffer size used in the dump, so you can omit this switch. If you specify a buffer size other than the one used for the dump, LOAD_II displays a warning.
/CONFIRM	Prompts for confirmation before trying to delete any file. Use this along with the /DELETE or /RECENT switches.
/DELETE or /D	Deletes any existing file with the same name (unless it is a directory file). If you use /V (/VERBOSE), LOAD_II will verify deletions. Permanent files cannot be deleted unless you also use the /DPERMANENT switch.
/DENSITY=n	Specifies the tape density. The density was established when the tape was dumped; you cannot change the density now. If you must use this switch, use /DENSITY=ADM to allow automatic density matching.
/DPERMANENT	Deletes any existing file with the same name even if the permanence attribute is on. Use with /DELETE or /RECENT.

/ELEMENTSIZE=value	Overrides the element size of files in the dump file. (The term file element size is defined in the CREATE command, /ELEMENTSIZE switch.) In AOS/VS, there is one type of file element. You can specify the size value as an integer between 1 and 65534. The default size is 4 blocks or a value chosen during AOS/VS system generation. You can force the default for all files loaded by using a value of −1. In AOS/VS II, there are two types of file element: primary and secondary. You can specify the size of each type, and the number of primary elements, using the form /ELEMENTSIZE=p:s:i The p indicates the primary element size, s the secondary element size, and i the number of primary elements. For example, the switch /ELEMENTSIZE=32768:4:8 would tell the utility to create all files in the dumpfile with a primary element size of 32768, a secondary element size of 4, and 8 primary elements. If you omit a value (including only a colon for the omitted value), the system uses the value from the dump file; for example, /ELEMENTSIZE=:8: specifies the stored dump file value for the primary size, 8 for the secondary size, and the dump file value for the number of primary elements. With either AOS/VS or AOS/VS II, if you want LOAD_II to create all files with the current LDU defaults, specify a size of −1. If you omit the /ELEMENTSIZE switch, LOAD_II will try to create the files with the element size in the dumpfile. If you use this switch to change the element size of files that will be used with shared page I/O, the new element size should be 4 or a multiple of 4. Otherwise, the ?SOPEN system call will fail. (By default, the Link utility creates program files with a primary and secondary element size of 32.)
/ERROR=pathname or /E=pathname	Writes error messages to the specified file. Error messages also go to the terminal or batch output file, and to the listing file. The file will be created if it does not exist; if it does exist, text will append to it.

/FASTFORWARD	Speeds up positioning an unlabeled tape on QIC, 4-mm DAT, and 8-mm cartridge tapes. (Other tape drives implement fast forward in hardware, so this switch has no effect with them.) Use this switch when specifying a large tape file number (for example, @MTJ0:150), which might otherwise cause a device time-out. You do not need to use the /MAXCAPACITY switch when using this switch; selecting this switch also sets MAXCAPACITY mode. If you specify a tape file number that does not actually exist, because this switch ignores logical end-of-tape marks, you receive an error when the tape positions beyond the last file on the tape. The computers that support this switch are ECLIPSE MV/4000 and above, excluding ECLIPSE MV/5000 DC series systems.
/FLAT	Loads all files directly into the working directory; does not retain any directory structure from the dump file.
/FOLLOWLINKS	Tries to follow links. If, while loading a non–link file, LOAD_II finds a link file with the same name, it tries to load the file into the resolution file specified by the link. (If there is no file with the same name in the directory, LOAD_II simply loads the file.) For example, it tries to load file SED.CLI into :UDD:ALICE, but finds a link file named SED.CLI already there. The link file specifies a resolution of :UTIL:SED:SED.CLI. The program will try to load SED.CLI into :UTIL:SED. This directory must already exist. Generally, for this switch to be useful, link files must already exist in the pertinent directories on disk.
/HASHFRAMESIZE=n	Sets a hashframe size of n for all directories (files of type CPD, DIR, and LDU) in the dump file. This switch has meaning only for AOS/VS; AOS/VS II ignores the switch.
/IBM	Loads from a tape that has an IBM–format label.
/INDEXELEMENTSIZE=value	(AOS/VS II only.) For each file, creates an index with an element size of value. To specify the LDU default, use –1.
/MAXCAPACITY	(Useful only with some cartridge units). Runs tape unit in streaming and buffered mode. Streaming can increase tape capacity and speed, but if the unit tries to stream and fails (you'll hear it starting and stopping), I/O will be slower than without this switch. Use this switch for large restore operations, when the system is otherwise idle.
/MEMORY=value	The switch has no effect. To maintain compatibility with AOS/VS Revision 7.63 and earlier, the switch accepts the following values: an integer 1 through 200 or the word MIN, LOW, MED, HIGH, or MAX.

/NACL	Does not load access control lists (ACLs) from the dump file. The program assigns the current default ACL to all files loaded (default ACL of the parent CLI process).
/NLOAD or /N	Does not load files; displays their names only. If you want to verify that a dumpfile is loadable, use this switch (not the commands COPY @NULL or XEQ DISPLAY) to read the dumpfile without loading files.
/NPERMANENCE	Does not retain permanence; loads files with permanence off.
/NPROMPT	Terminates the load operation if the utility encounters a situation that requires interaction. If an error occurs that normally produces an interactive prompt, the utility terminates and forces you to restart the load rather than accepting any recovery action. (EXEC still allows normal operator intervention with labeled tapes.)
/NSPAN	(Unlabeled tapes only.) Does not span more than one reel of tape. Aborts the load when it needs to use a second unlabeled volume.
/OWNER=name	Verifies that the tape belongs to the owner you specify with name. The utility terminates with an error message when the name field from the tape does not match the specified name.
/RECENT	Loads only those files created more recently than existing files of the same name, unless the existing file is a directory. (Comparison is based on the time created, not the time last modified.)
/SEQUENTIAL	For labeled tape that has been mounted via MOUNT/VOLID= and the EXEC command MOUNTED. Prevents the program from rewinding the tape after completing the load. This saves rewind and spool forward time if you want to load again from the same tape volume.
/SPECIFIC	Starts loading from a specific volume in the middle of the dumpfile and ignores "Indecipherable dump format" errors. LOAD_II continues to the next file and starts the load from there. For labeled tape, the LOAD_II command specifies the volume via @LMT:volid:tape–filename. For unlabeled tape, LOAD_II loads from the tape specified in the command line (for example, MTJ1:0). Use this switch to load from a logical file within a file set without going through all preceding volumes. This switch is further described in Chapter 6.

/STATISTICS	Writes load statistics to your terminal or the listing file.
/TRAVERSE=directory–typecode	Specifies the directory types to traverse (go through) while searching for files to load. Directory typecodes include LDU (logical disk unit), CPD (control point directory), and DIR (standard directory). The utility finds files that exist only in these specified directory types.
/TRAVERSE=\directory–typecode	Goes through all directories *except* those of directory–typecode searching for files to load. Directory typecodes include LDU (logical disk unit), CPD (control point directory), and DIR (standard directory). The utility finds files that exist only in directory types not excluded with this switch.
/TYPE=typecode or /T=typecode	Loads files of type code only, where the typecode value is one of the following forms (listed in Table 2–8):

xxx	a 3-letter mnemonic (such as DIR or CPD for a directory, LNK for a link file)
n	a decimal number (0–255) that defines a type code. Valid file types are system–defined types 0, 10–13, 64–78, 81–103, and 105–127, or user–defined types 128–255 (see Table 2–8).
m–n	numbers that select a range of file types.

/TYPE=\typecode or /T=\typecode	Loads files *except* those of type code, where type code is one of the code types listed in the left column above. You can use more than one /TYPE= switch in a command.
/UPDATE	Updates directories with any new information from the dumpfile, including CPD maximum size, access control list, permanence, and user data areas (important for INFOS II files). Regardless of the switches you use, if a file already exists and is a directory, the program will not delete the directory but will use the existing directory.
/VERBOSE or /V	Displays the names of file(s) loaded on your terminal screen, or, if you also include the /LISTING=pathname switch, to the specified file. To display names *and* write them to the list file, use the /BOTH= switch.

LOAD_II Example 1

```
) LOAD_II/V  @MTB0:0⤶
```

This command loads all files from the dump file in file 0 of the tape on unit MTB0 into the working directory. It maintains the directory structure (if any) from the dump file. The /V switch tells the utility to display the filenames loaded on the terminal.

In this example, and all others following, if the dump file spans more than one volume, the program will ask for another volume:

Mount the next volume, volume: n
Respond MOUNTED <device> or REFUSED when ready.

The person who issued the command can then mount another tape and type MOUNTED or, if using a different unit, MOUNTED unitname. (The utility always prompts for a volume number (*n*) with unlabeled tape. It cannot check the volume mounted, so don't confirm by typing MOUNTED until you've actually mounted another tape.)

LOAD_II Example 2

```
) LOAD_II/RECENT/BOTH=LOAD.LIST/V @MRCTAPE000A00:0  +.F77⤶
```

This command loads FORTRAN 77 source files from the first tape file of the tape on unit MRCTAPE000A00 into the working directory. If a file in the dump file has the same name as a file on disk, and was created more recently than the one on disk, the program deletes the version on disk and replaces it with the newer one.

The /BOTH and /V switches tell the program to display dumped file names on the terminal and write them to file LOAD.LIST in the working directory.

LOAD_II Example 3

```
) LOAD_II/MAXCAPACITY/BOTH=LOAD.LIST/V/RECENT  @MTJ0:0  +.F77⤶
```

This command has the same effect as the previous one, but with switches that let the program run the cartridge tape on MTJ0 more efficiently.

LOAD_II Example 4

```
) LOAD_II/V/AFTER/TLM=21-MAY-89:12/TYPE=\LNK  @MTJ0:2  MYDIR:#⤶
```

This command loads from the third file of the tape on unit MTJ0. The program loads all files created or modified on or after noon, May 21, 1989, excluding links, from directory MYDIR.

LOAD_II Example 5

```
) SUPERUSER ON ↵
Su) MOUNT/VOLID=V1/VOLID=V2/VOLID=V3  MYTAPE  Ready to restore↵
```

(The system operator mounts the first tape.)

```
Su) DIR : ↵
Su) LOAD_II/V/BOTH=:UDD:CHRIS:FILES_RESTORED/RECENT  & ↵
&) :UDD:[!USERNAME]:MYTAPE:FILESET1 UDD:CHRIS:PROJECTS:# ↵
```

... (System reads from multiple volumes and loads files) ...

```
Su) DISMOUNT MYTAPE  Restoration is done. ↵
```

This shows the restoration of directory :UDD:CHRIS:PROJECTS and its files from a system backup done on labeled tape. The MOUNT command asks the system operator (person at the system console) to mount tape volume V1 (first of a sequence of three volumes). The DIRECTORY command makes the root directory the working directory since the original backup was done from the root directory. The LOAD_II command searches the dump file named FILESET1, via tape linkname MYTAPE (in user's initial directory), for the specified directory and loads it. A listing of restored files goes to file FILES_RESTORED in CHRIS' directory. After the load is done, the DISMOUNT command asks the operator to dismount the last tape.

LOCK *Command*

Locks the CLI (CLI32 version — CLI16 version precedes).

Format

```
LOCK  ┌CLI-command-to-disable       ┐
      └/CX EXEC-command-to-disable  ┘
```

A locked CLI will not execute certain commands. It can help safeguard system console (or any terminal) from unauthorized or accidental use, misuse, or security breach.

If you include one or more command arguments, the CLI will disable those commands only. If you omit arguments, the CLI disables all CLI commands that relate to security, including the following:

BLOCK	DEBUG	JPRELEASE	QFTA	SUPERUSER
BYE	DELETE	MOVE	QSUBMIT	TERMINATE
CHAIN	EXECUTE	PRIVILEGE	RELEASE	XEQ
CONNECT	INITIALIZE	PROCESS	RENAME	
COPY	JPINITIALIZE	QBATCH	SUPERPROCESS	

NOTE: The DUMP and LOAD commands do not work with a locked CLI since they invoke the PROCESS command, which is locked.

Thus, you cannot execute programs from a CLI that was locked via LOCK without an argument.

When you include the /CX switch without arguments, the CLI disables all EXEC commands (explained in the EXEC chapter, *Managing AOS/VS and AOS/VS II*). If you include arguments, the CLI locks those EXEC commands only; for example, LOCK/CX HALT locks the EXEC HALT command. Locking EXEC commands is most useful for the master CLI that runs on the system console.

When you lock it, the CLI automatically turns off Superuser, Superprocess, and/or System Manager privilege if these were on, and ignores the process termination sequences CTRL–C CTRL–B, CTRL–C CTRL–E, and CTRL–D CTRL–D.

The CLI32 LOCK command requires you to specify a password. If no password has been defined when you type LOCK, the CLI will notify you: *No password is in effect.* To define a password, type the command PASSWORD; then specify a password which can be 1 through 32 characters long. The CLI asks you to verify the password by typing it a second time. After the CLI accepts the password, retry the LOCK command.

If a password *has* been defined when you type LOCK, the CLI will prompt for the password; the CLI will then obey the LOCK command after you supply the password.

After you define a password, you can save it (encrypted) in a file using the command form PASSWORD/WRITE=password–file. Then, whenever you run a new CLI, you can re-establish the original password and lock the CLI via the command forms

PASSWORD/READ=password–file
LOCK/FILE=password–file.

- No arguments.
- No templates.
- Requirement: *Standard* (but see note below).
- See also: PASSWORD, UNLOCK.

Why Use It?

Use the LOCK command at the system console (or any terminal) to prevent the unauthorized use of system resources, and to protect the system and its information from security breaches, accidents, or malicious destruction.

You can include PASSWORD and LOCK commands in the system UP.CLI macro to lock the CLI automatically. If you want to lock commands on your user terminal, you can include these commands in your log-on macro.

CAUTION: If you forget the password, you cannot exit from a locked CLI. You will need to terminate it (or have it terminated) from a superior process. For this reason, be especially careful when locking PID 2.

Use PASSWORD/READ= and LOCK/FILE= only to retrieve a password saved in a file created by PASSWORD/WRITE=.

Command Switches

The section "Universal CLI Switches," earlier in this chapter, describes the switches /1, /2, /L, /L=pathname, and /Q, which you can use with all commands. That section also explains the CLI32 switches /STR= and /ESTR=.

/ALL	Disables (locks) all CLI commands except UNLOCK. To re-enable all commands, you must use UNLOCK/ALL.
/CX	Disables (locks) commands to EXEC (CONTROL @EXEC commands). If you include one or more EXEC commands as arguments, the CLI locks only those commands. By default, all EXEC commands are unlocked when you lock the CLI without using this switch.
/FILE=pathname	Reads the password, in encrypted form, from the file pathname. Earlier you must have written the password to the pathname file using a PASSWORD/WRITE=pathname command. You can use a LOCK/FILE=pathname command in your UP macro or your log-on macro to lock the CLI automatically.

LOCK (CLI32) Command Switches (continued)

/STATUS	(CLI32 only.) Without an argument, displays the names of all currently locked non-EXEC commands. To determine which EXEC commands are locked, use this switch with /CX. When you supply an argument, displays the names of those specified command(s) that are locked. Does not require password.
/V	Displays the names of the commands you are locking. Requires password.
/VERIFY	(CLI32 only.) Same as /V.

LOCK (CLI32) Example 1

```
) LOCK⤶
Warning: No password is in effect
) PASSWORD⤶
Password: JOAN⤶                    (Password does not echo.)
Confirm password: joan⤶            (Confirmation does not echo.)
) LOCK⤶
Password: JOAN⤶
) LOCK/STATUS SUPERUSER⤶           (Verify the lock status of SUPERUSER.)
SUPERUSER                          (Display indicates SUPERUSER is locked;
) SUPERUSER⤶                       therefore, the command will not execute.)
Error: Command is locked, SUPERUSER
```

In this sequence, a person tries to lock the CLI, fails because no password has been defined, defines a password, locks the CLI, and then tests a locked command.

The person wants to use the same password later, so saves it to file PW_FILE.

```
) PASSWORD/WRITE=PW_FILE⤶
```

Later, he or she can lock a new CLI using the commands:

```
) PASSWORD/READ=PW_FILE⤶
) LOCK/FILE=PW_FILE⤶
```

then unlock the CLI with the original password, and test the status of a locked command:

```
) UNLOCK⤶
Password: joan⤶
) LOCK/STATUS SUPERUSER ⤶          (Verify the lock status of SUPERUSER.)
)                                  (No response indicates that it is not locked.)
)SUPERUSER ⤶                       (The command successfully executes.)
Su)
```

■

LOCK (CLI32) Example 2

```
) LOCK/CX HALT⤶                    (Lock the EXEC HALT command.
Password: XYZ⤶                     (Password does not echo.)
) LOCK/CX/STATUS                   (Display the locked EXEC commands.)
CONTROL @EXEC HALT
```

LOGEVENT *Command*

Writes a message into the system log file (SYSLOG).

Format

LOGEVENT message

This command enters the specified message, as a text string, into the system log file, :SYSLOG. However, you must have the Superuser privilege, have it turned on, and have system logging in progress.

To enter lengthy messages, use several LOGEVENT commands.

The REPORT utility formats and displays records from :SYSLOG. For information about the REPORT utility, see *Managing AOS/VS and AOS/VS II.*

- No templates.
- No argument switches.
- Requirement: *Superuser.*
- See also: SYSLOG and REPORT (to generate a report from log file records). These utilities are further described in *Managing AOS/VS and AOS/VS II.*

Why Use It?

Use the LOGEVENT command to make an entry in the system log file (SYSLOG). You can enter comments or other notations that may help to explain the contents of the log file.

Command Switches

The section "Universal CLI Switches," earlier in this chapter, describes switches /1, /2, /L, /L=pathname, and /Q, which you can use with all commands. That section also explains the CLI32 switches /STR= and /ESTR=.

LOGEVENT Example

```
) SUPERUSER ON)
Su) LOGEVENT NEW PERIPHERALS INSTALLED ON 17 JANUARY.)
```

The LOGEVENT command writes a message to the system log file.

!LOGON *Pseudomacro*

Expands to CONSOLE, BATCH, or null, depending on how you logged on the CLI.

Format

[!LOGON]

This pseudomacro returns the mode in which the CLI is running. If you logged on under the EXEC process, the pseudomacro returns either CONSOLE or BATCH. If you logged on under a process other than EXEC, the pseudomacro returns a null value.

- No arguments.
- No macro name switches.
- Requirement: *Standard.*

Why Use It?

You can use this pseudomacro within a macro to determine whether or not the calling CLI is an interactive session or a batch job.

!LOGON Example

(within a macro)

```
[!equal, [!logon], BATCH]
            write This CLI is running in batch.
            ...
[!else]
            write This CLI is running on a terminal.
            ...
[!end]
```

This macro determines if the user is running in batch mode or not. If so, the CLI executes the statements up to the !ELSE pseudomacro; if not, it executes the statements between !ELSE and !END.

!LOOPEND

Pseudomacro

Ends a CLI32 loop sequence (CLI32 only).

Format

[!LOOPSTART]

.. commands to execute (optionally located here)

    [!EXIT/LOOP] (optional exit from loop)

.. commands to execute (optionally located here)

[!LOOPEND]

The !LOOPEND pseudomacro ends the sequence of CLI commands introduced with !LOOPSTART. You must use !LOOPEND to indicate the end of a CLI32 loop sequence. At the !LOOPEND statement, the CLI examines the loop's iteration count, if one has been specified. The CLI either exits the loop if a specified count has been satisfied, or begins another iteration. An optional !EXIT/LOOP pseudomacro provides another way to exit a loop.

- No templates.
- No argument switches.
- No macro name switches.
- Requirement: *Standard.*
- See also: !LOOPSTART and !EXIT.

Why Use It?

Use the !LOOPEND pseudomacro to end the sequence of CLI commands introduced with !LOOPSTART. If you execute a macro that does not have a !LOOPEND statement for each loop sequence, the CLI displays

Error: Missing !LOOPEND

For more information on CLI32 loop pseudomacros, see Chapter 4.

!LOOPEND Example 1

The following macro named FOUR_WORDS.CLI executes a loop four times.

```
STRING/NAME=new_word/K; STRING/NAME=all_words/K
[!LOOPSTART 4]
STRING/NAME=new_word [!READ Type a word and Enter: ]
STRING/NAME=all_words &
            [!STRING/NAME=all_words] [!STRING/NAME=new_word]
[!LOOPEND]
WRITE  [!STRING/NAME=all_words]
```

```
) FOUR_WORDS ⤶
Type a word and Enter: Now ⤶
Type a word and Enter: is ⤶
Type a word and Enter: the ⤶
Type a word and Enter: time ⤶
 Now is the time
```

!LOOPEND Example 2

The following macro named SOME_WORDS.CLI executes a loop until either you respond to the prompt by pressing Enter without first typing a word, or you type (and Enter) the fourth word.

```
STRING/NAME=new_word/K; STRING/NAME=all_words/K
[!LOOPSTART 4]
STRING/NAME=new_word [!READ Type a word and Enter or just Enter: ]
     [!EQUAL [!STRING/NAME=new_word],]
            [!EXIT/LOOP]
     [!END]
STRING/NAME=all_words &
                    [!STRING/NAME=all_words] [!STRING/NAME=new_word]
[!LOOPEND]
WRITE  [!STRING/NAME=all_words]
```

```
) SOME_WORDS ⤶
Type a word and Enter or just Enter: Not ⤶
Type a word and Enter or just Enter: now ⤶
Type a word and Enter or just Enter:  ⤶
 Not now
```

!LOOPSTART *Pseudomacro*

Begins a CLI32 loop sequence (CLI32 only).

Format

```
[!LOOPSTART [iteration_count]]
.. commands to execute (optionally located here)
    [!EXIT/LOOP] (optional exit from loop)
.. commands to execute (optionally located here)
[!LOOPEND]
```

The !LOOPSTART pseudomacro begins the sequence of CLI commands that ends with !LOOPEND. The loop executes until an optional *iteration_count* is satisfied, an optional !EXIT/LOOP pseudomacro is reached, or you press CTRL–C, CTRL–A. At the !LOOPEND statement, the CLI examines the loop's iteration count, if one has been specified. The CLI either exits the loop if a specified count has been satisfied, or begins another iteration.

Valid iteration count values range from 0 through 4,294,967,295, which is $2^{32} - 1$. Commands within the loop do not execute if you use an iteration count of 0.

Loops can be nested: one !LOOPSTART, one !LOOPEND, and any number of !EXIT/LOOP pseudomacros per level. Loops are not allowed to span macros and conditional pseudomacros. The following sequence would generate an error.

```
[!EQUAL,%1%,%2%]
            [!LOOPSTART 10]
                commands
[!END]
            [!LOOPEND]
```

- No templates.
- No argument switches.
- No macro name switches.
- Requirement: *Standard.*
- See also: !LOOPEND and !EXIT.

Why Use It?

Use the !LOOPSTART pseudomacro in conjunction with the !LOOPEND pseudomacro to define a sequence of CLI commands to be executed a specific number of times, until a condition is recognized, or indefinitely.

For more information on CLI32 loop pseudomacros, see Chapter 4.

!LOOPSTART Example 1

In the following macro named M2N.CLI, the second argument is used to determine the number of times the loop executes.

```
\\ This macro raises the integer value in the first argument
\\ to the integer power in the second argument.
\\
VAR0 %1%
[!LOOPSTART [!USUBTRACT %2% 1]]
VAR0 [!UMUL [!VAR0] %1%]
[!LOOPEND]
WRITE %1%^%2% = [!VAR0]
```

```
) M2N 9 3 ↵
9^3 = 729
```

!LOOPSTART Example 2

The following macro named MAKESTRINGS.CLI executes a loop a maximum of five times — fewer if the exit condition is met.

```
\\ This macro accepts as many as five file names – storing each
\\ one in a separate CLI string

[!LOOPSTART 5]
STRING/NAME=[!VAR0] [!READ Type name or none and Enter: ]
     [!EQUAL [!STRING/NAME=[!VAR0]],none
             STRING/NAME=[!VAR0]/KILL
             [!EXIT/LOOP]
     [!END]
WRITE String [!VAR0] contains "[!STRING/NAME=[!VAR0]]"
VAR0 [!UADD [!VAR0] 1]
[!LOOPEND]
WRITE Done – [!VAR0] names entered
```

```
) MAKESTRINGS ↵
Type name or none and Enter: FILE1 ↵
String 0 contains "FILE1"
Type name or none and Enter: FILE2 ↵
String 1 contains "FILE2"
Type name or none and Enter: FILE3 ↵
String 2 contains "FILE3"
Type name or none and Enter: NONE ↵
Done – 3 names entered
```

!LOOPSTART Example 3

The following macro named LISTSTRINGS.CLI executes a loop as long as it finds existing string variables.

```
\\ This macro lists as many as five previously defined strings
\\ with their contents.

VAR0      0
[!LOOPSTART ]                       \\ Loop indefinitely
     [!EQUAL [!LENGTH [!STRING/NAME=[!VAR0]]], 0]
            [!EXIT/LOOP]            \\ No more strings defined
     [!END]
WRITE String [!VAR0] contains "[!STRING/NAME=[!VAR0]]"
VAR0 [!UADD [!VAR0] 1]              \\ Increment string count
[!LOOPEND]
WRITE Done – [!VAR0] strings listed
```

```
) LISTSTRINGS ↲
String 0 contains "FILE1"
String 1 contains "FILE2"
String 2 contains "FILE3"
Done – 3 strings listed
```

MESSAGE *Command*

Displays the text message that corresponds to an error code.

Format

MESSAGE errorcode *[...]*

This command displays the text message that corresponds to an error code, as defined in the system error code file, ERMES. Building the ERMES file is explained in the "Installing" manual for your operating system.

- No templates.
- No argument switches.
- Requirement: *Standard*.

Why Use It?

Use the MESSAGE command to find the error message that corresponds to a specific error code.

NOTE: By default, the CLI interprets the error code you specify as an octal value. If the error code is decimal, include the /D switch.

Command Switches

The section "Universal CLI Switches," earlier in this chapter, describes switches /1, /2, /L, /L=pathname, and /Q, which you can use with all commands. That section also explains the CLI32 switches /STR= and /ESTR=.

/D	Interprets error code arguments as decimal, not octal values.
/DECIMAL	(CLI32 only.) Same as /D.

MESSAGE Examples

```
) MESSAGE 25↵
25 File does not exist
```

This command displays the message text associated with the octal error code 25.

```
) MESSAGE 26↵
26 Filename already exists

) MESSAGE/D 22↵
22 Filename already exists
```

The first command displays the error message for octal code 26. The second command uses the /D switch to display the message for the decimal equivalent.

MIRROR *Command*

Initializes the other image of a mirrored Logical Disk Unit (LDU), and begins to synchronize the images. (AOS/VS version — AOS/VS II version follows.)

Format

MIRROR pathname-of-initialized-image unitname *[unitname]* ...

where:

pathname-of-initialized-image	Consists of a full pathname. For example, if an LDU named UDD1 was initialized in the root, its pathname is :UDD1.
unitname	Specifies a disk unit containing an LDU image that has the same filename and same size — created with the Disk Formatter (DFMTR) program — as the initialized image.

The MIRROR command initializes the other image of a mirrored LDU and begins to synchronize the images. (To initialize the first image of a mirrored LDU, use the CLI command INITIALIZE/NOMIRROR.)

CAUTION: Mirroring overwrites all information on the mirrored LDU.

- No templates.
- No argument switches.
- Requirement: You must have Owner access to the LDU root directory and Read access to the :PER entries for each unitname, or have Superuser on.
- See also: INITIALIZE and the manual *Managing AOS/VS and AOS/VS II.*

Why Use It?

LDU mirroring offers higher availability by providing a copy of an LDU. If something happens to one of the images, AOS/VS will break the mirror and maintain access to the other image — providing continuous access to information on the LDU.

Use MIRROR with unsynchronized images to start synchronizing an initialized LDU to its mirror image. (If the images were released normally — for example, by system shutdown — they remain synchronized and you can initialize and start mirroring on them with the INITIALIZE command, form unit!unit. Refer to the manual *Managing AOS/VS and AOS/VS II* for a detailed discussion of mirroring.

Command Switches

The section "Universal CLI Switches," earlier in this chapter, describes switches /1, /2, /L, /L=pathname, and /Q, which you can use with all commands. That section also explains the CLI32 /STR= and /ESTR= switches.

MIRROR, AOS/VS Version

/FORCESYNC	Forces the system to synchronize based on the older image. (The older image is the initialized image.) You must include this switch to mirror an older LDU image to a newer one. This is an unusual form of the command, since generally you synchronize the newer image to the older one (/SYNC switch). Be sure that the older image is the one you prefer.
/SYNC	Tells the system to synchronize based on the newer image. (The newer image is the initialized image.) Include this switch to mirror a newer LDU image to an older one. Since MIRROR/SYNC retains the information on the newer LDU, it is the normal usage of the command.
/WAIT	Tells the system to suspend your CLI until synchronization is complete. Depending on image size, this may take minutes or hours.

MIRROR (AOS/VS) Example

```
Su) INITIALIZE/NOMIRROR @DPJ1 ↵
UDD1
Su) MIRROR/SYNC :UDD1 @DPJ2 ↵
```

The first command initializes the LDU image in DPJ1; the system displays the LDU name, which is UDD1. The second command starts synchronizing the image in DPJ2 with the initialized image.

MIRROR *Command*

Starts to synchronize (mirror) an LDU image with another initialized image, or breaks a mirror image. (AOS/VS II version — AOS/VS version precedes.)

Format

$$\text{MIRROR pathname-of-initialized-image} \left\{ \begin{array}{l} \text{unitname } \textit{[unitname ...]} \\ \text{unique\_ID/unitname}\textit{[/unitname ...]} \end{array} \right\}$$

where:

pathname-of-initialized-image	Consists of a full pathname. For example, if an LDU named UDD1 was initialized in the root, its pathname is :UDD1.
unitname	Specifies a disk unit containing an LDU image that has the same filename and same size (created with the Disk Jockey program) as the initialized image.
unique_ID	Combines the LDU filename with an image extension such that the complete name is unique to the system, not just to a disk unit or LDU. For example, UDD1.IMAGE1 on disk unit DPJ1 mirrored as UDD1.IMAGE2 on unit DPJ2.

The MIRROR command starts synchronizing (mirroring) an LDU image with another image, or breaks a mirror image. The LDU image you want to use as a source must be initialized (INITIALIZE/NOMIRROR command). Use the LDUINFO utility to find the more recent image.

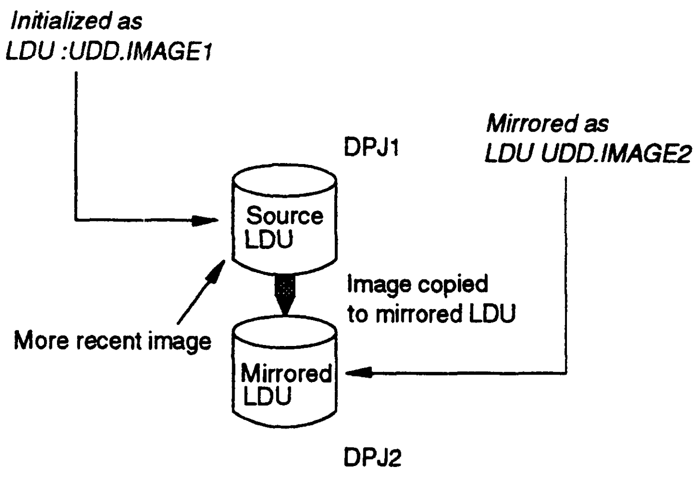
CAUTION: Mirroring overwrites all information on the mirrored LDU.

The unitname must contain an LDU image that has the same filename and same size (created with the Disk Jockey program) as the initialized image. As the pathname-of-the-initialized-image, use a full pathname; for example, if an LDU named UDD1 as initialized in the root, its pathname is :UDD1.

Assume that the newer image is on DPJ1 and an older image is on DPJ2. To synchronize based on the *newer* image, initialize the newer image and then use MIRROR with the /SYNC switch.

For example, initialize a newer image on DPJ1 and mirror it on DPJ2:

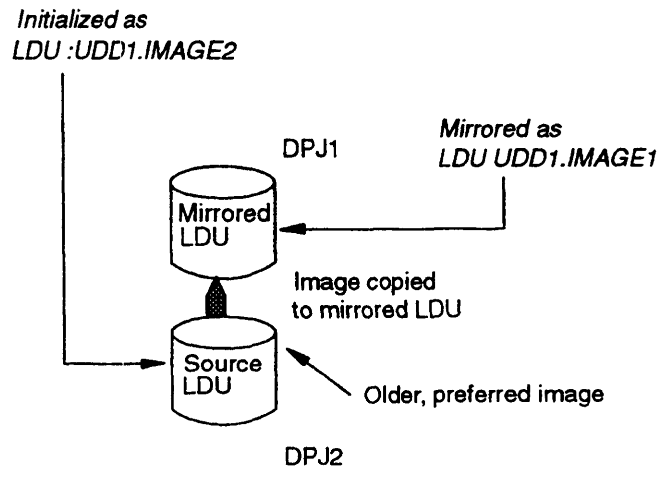
```
                                              (Initializes the newer
Su) INITIALIZE/NOMIRROR  @DPJ1 ⤶              source image)
UDD                                           (System echoes LDU filename)
Su) MIRROR/SYNC :UDD UDD.IMAGE2/@DPJ2 ⤶       (Starts synchronizing the
                                              image on DPJ2)
```



Assume either that the newer image has been corrupted, or for some other reason you want to discard it and retain the older image. To synchronize based on the *older* image, initialize the older image and then use MIRROR with the /FORCESYNC switch.

For example, initialize an older image on DPJ2 and mirror it on DPJ1:

```
                                                 (Initializes the older
Su) INITIALIZE/NOMIRROR  @DPJ2 ⤶                 source image)
UDD                                              (System echoes LDU filename)
Su) MIRROR/FORCESYNC :UDD UDD.IMAGE1/@DPJ1 ⤶     (Start synchronizing
                                                 the image on DPJ1)
```



MIRROR, AOS/VS II Version (continued)

To remove an LDU image from a mirrored set (stop mirroring), you can use the /BREAK switch; see comments under /BREAK below.

If there is more than one LDU on a physical disk, you must specify the LDU unique ID as well as the unit name. Use the form

MIRROR pathname–of–initialized–image unique_ID/unitname*[/unitname ...]*

For example, assume that you want to mirror an LDU named XDATA whose images exist on units DPJ11 and DPJ12. The unique ID on DPJ11 is XDATA.IMAGE1; on DPJ12 it is XDATA.IMAGE2. Each disk holds more than one LDU. The image on unit DPJ11 was initialized in the root directory. To mirror on the LDU on DPJ12, type

```
Su) MIRROR/SYNC :XDATA  XDATA.IMAGE2/@DPJ12 ↵
```

If this LDU spanned two disks, in DPJ12 and DPJ13, you would type

```
Su) MIRROR/SYNC :XDATA  XDATA.IMAGE2/@DPJ12/@DPJ13 ↵
```

You can learn which LDUs are on which physical disks using the LDUINFO utility or Disk Jockey, View LDU Information screen, using keyword LDINFO.

By default, the system will use hardware mirroring if possible. Otherwise it will use software mirroring (which may take longer). You can force software mirroring with the /NOHARDWARE switch. The system will confirm your command with a *software mirrored* or *hardware mirrored* status message.

Depending on image size/type, synchronization may take hours. The system will display a *Synchronization ... complete* message when it is done.

- No templates.
- No argument switches.
- Requirement: You must have Owner access to the LDU root directory and Read access to the :PER entries for each unitname, or have Superuser on.
- See also: INITIALIZE, LDUINFO (also in *Managing AOS/VS and AOS/VS II*).

Why Use It?

LDU mirroring offers higher availability by providing a copy of an LDU. If something happens to one of the images, AOS/VS II will break the mirror and maintain access to the other image — providing continuous access to information on the LDU.

Use MIRROR with unsynchronized images to start synchronizing an initialized LDU to its mirror image. (If the images were released normally — for example, by system shutdown — they remain synchronized and you can initialize and start mirroring on them with the INITIALIZE command, form unit!unit.)

The system will use hardware mirroring if possible. Otherwise it will use software mirroring (which may take longer). You can force software mirroring with the /NOHARDWARE switch. The system will confirm your MIRROR command with a *software mirrored* or *hardware mirrored* status message.

Mirroring is further described in the "Installing" manual for your operating system and in *Managing AOS/VS and AOS/VS II.*

MIRROR, AOS/VS II Version Command Switches

The section "Universal CLI Switches," earlier in this chapter, describes switches /1, /2, /L, /L=pathname, and /Q, which you can use with all commands. That section also explains the CLI32 /STR= and /ESTR= switches.

/BREAK	Tells the system to remove the specified LDU image from the mirrored set. Use this when you want to remove a disk from use. MIRROR/BREAK is meant to break a mirror for such things as disk diagnostics or maintenance; it does not release the LDU image normally. If possible, ensure that the LDU is in a known state before issuing MIRROR/BREAK to it; for example, if CEO is running on the LDU, shut down CEO before issuing the command.
/FORCESYNC	Forces the system to synchronize based on the older image. (The older image is the initialized image.) You must include this switch to mirror an older LDU image to a newer one. This is an unusual form of the command, since generally you synchronize the newer image to the older one (/SYNC switch). Be sure that the older image is the one you prefer.
/NOHARDWARE	Forces the system to use software mirroring (the operating system, not the disk controller, does the mirroring I/O). By default, if you omit this switch, the system tries to use hardware mirroring; then, if hardware mirroring is not possible, it tries to use software mirroring. Use this switch if you want software mirroring even if hardware mirroring is possible.
/SYNC	Tells the system to synchronize based on the newer image. (The newer image is the initialized image.) Include this switch to mirror a newer LDU image to an older one. Since MIRROR/SYNC retains the information on the newer LDU, it is the normal usage of the command.
/TRESPASS	Starts synchronizing the LDU even though it is marked as owned by another system. (The disk was last initialized by another system.) This switch is designed for systems that use multiported (shared) disks. Use the switch only if you're sure the other system has failed and your system needs access to the multiported disk information. If the other system is still running and you use this switch, disk information will be corrupted.
/WAIT	Tells the system to suspend your CLI until synchronization is complete. Depending on image size, this may take minutes or hours.

MIRROR, AOS/VS II Version Example 1

This example shows initialization and synchronization of an LDU named UDD1 in units DPJ1 and DPJ2. The image in DPJ1 is the primary image, since it's initialized first.

```
) SUPERUSER ON ; DIRECTORY : ↵         (Turns Superuser on and makes the
                                        root (:) the working directory.)

Su) INITIALIZE/NOMIRROR @DPJ1 ↵         (Initializes one image. /NOMIRROR is
                                        needed to initialize one image of a
                                        mirrored set.)
UDD1                                    (System displays LDU name.)

Su) MIRROR/SYNC :UDD1 @DPJ2 ↵           (Starts synchronizing image in DPJ2.
                                        (System displays mirror status.)
From system:
LDU 'UDD1', image 'UDD1.IMAGE2' is hardware mirrored -- unsynchronized
Su)
...                                     (Time passes while synchronization
                                        completes.)

Synchronization of mirrored LDU in unit DPJ2 is complete.
```

MIRROR, AOS/VS II Version Example 2

The next example shows initialization, desynchronization, and synchronization of an image. The primary image is in DPJ1, the secondary in DPJ2. The unique ID of the secondary image is UDD1.IMAGE2.

```
) SUPERUSER ON ; DIRECTORY : ↵          (Turns Superuser on and makes the
                                        root (:) the working directory.)

Su) INITIALIZE @DPJ1!@DPJ2 ↵            (Initializes both images. For this to
                                        work, the images must have been
                                        synchronized when released.)
...                                     (Message on mirror status.)
...                                     (System runs; images synchronized.)

Su) MIRROR/BREAK :UDD1 UDD1.IMAGE2 ↵    (Stops mirroring on
                                        UDD1.IMAGE2 — perhaps for
                                        a diagnostic check.)
...                                     (System runs, images desynchronized)
Su) MIRROR/SYNC :UDD1 @DPJ2 ↵           (Starts synchronizing DPJ2 with
                                        DPJ1.)
```

MIRROR, AOS/VS II Version Example 3

For another example, suppose there are multiple LDUs on the disks in units DPJ11 and DPJ12. You want to mirror the LDU named XXX which exists on both units. Its unique ID on DPJ11 is XXX.IMAGE1; on DPJ12 the unique ID is XXX.IMAGE2. The image on DPJ11 has already been initialized (thus is the primary image) in the root directory. The following command starts mirroring on the image in DPJ12:

```
Su) MIRROR/SYNC :XXX XXX.IMAGE2/@DPJ12 ↵     (Start mirroring to DPJ12.)
                                             (System confirms with status.)
```

From system:
LDU 'XXX', image 'XXX.IMAGE2' is hardware mirrored -- unsynchronized

MOUNT *Command*

Asks the system operator to mount a tape on tape unit.

Format

MOUNT linkname message

This command requests the system operator to place a magnetic tape on a unit for your exclusive use. You must be logged on under EXEC, either at a terminal or in a batch job submitted with the QBATCH or QSUBMIT command.

The linkname argument is a filename that you use for tape access. It can be any valid filename and need have no relation to any file on the tape(s). This linkname is created in your initial user directory (form :UDD:username:linkname). It cannot already exist in your initial working directory. You must use the linkname — not the tape device name — for access. When you use the DISMOUNT command, the system deletes the link and releases the tape drive from your exclusive use.

The message argument is any text (up to 80 characters) you want displayed on the system console. Use this argument to enable the system operator to find the appropriate tape. For example, you might type

```
) MOUNT MYTAPE Please mount a scratch tape — ring in. ⤶
```

The system sends the request to the system console, where someone acting as system operator can respond to it. If the operator mounts the tape on the unit, the system creates the linkname in your initial working directory (unless you specify otherwise with the /DIRECTORY switch).

To ask the operator to mount a labeled tape, include the /VOLID= switch to indicate the volume ID(s) you want.

If you queue a batch job that uses the MOUNT command, append the /OPERATOR switch to the QBATCH command. This guarantees that the system will not start the job until an operator is on duty.

By default, after you type a MOUNT command, the CLI prompt doesn't return to your terminal until someone at the system console types a response to your request. If this happens and you want to skip the delay, abort the command with CTRL–A CTRL–C. You can avoid this delay and have the prompt return to your terminal immediately by using the MOUNT /NOPEND switch.

After your MOUNT command, the system posts your request in the mount queue (MOUNTQ). The request will remain in this queue until you issue the DISMOUNT command or the QCANCEL command against the request sequence number. You can check the status of the MOUNTQ queue with the QDISPLAY command.

Using MOUNT is further described in Chapter 6.

MOUNT (continued)

- No templates.
- No argument switches.
- Requirement: *Standard* (for sons of EXEC only).
- See also: DUMP or DUMP_II, LOAD or LOAD_II, DISMOUNT (to remove a mounted tape), OPERATOR (to work with labeled diskettes), LABEL (to label a tape or diskette).

System Operator's Role

To fulfill a mount request, someone must be on duty (as operator) to mount tapes. To tell EXEC that an operator is on duty, someone at the system console (or logged on as OP) must issue the command

```
) CONTROL @EXEC OPERATOR ON ↲
```

If this command has not been issued, prompt messages will not be displayed on the system console and mount requests may wait indefinitely in the mount queue. Generally, it is fruitless to issue a mount request if an operator is not on duty. You can tell whether an operator is on duty by typing

```
) WRITE [!OPERATOR] ↲
```

The answer will be ON or OFF.

If you mount your own tapes and perform your own loads and dumps, you have access to the system console and act as your own system operator (using these CONTROL @EXEC commands).

When you answer a mount request by mounting a tape, be sure the tape is the correct one. EXEC does not check until the user tries to read or write the tape. You can discover the contents of a tape label, including the volume ID and fileset–name, with the TYPE command. An example is

```
) TYPE @MTB0:0 ↲
```

You can label tapes with the LABEL utility. Labels are detailed in Chapter 6.

Why Use It?

Use the MOUNT command if a system operator controls the mounting and dismounting of magnetic tapes on your system, or if you want to use labeled tapes. Even if you mount tapes yourself, you must use MOUNT with the /VOLID switch if you want to use labeled tapes.

MOUNT is useful with tape only; for labeled diskettes, see the OPERATOR command.

MOUNT Command Switches

The section "Universal CLI Switches," earlier in this chapter, describes switches /1, /2, /L, /L=pathname, and /Q, which you can use with all commands. That section also explains the CLI32 switches /STR= and /ESTR=.

/AFTER=date:time

Processes this request after the specified date and time, which appear in the following form: dd–mon–yy:hh:mm:ss. This switch effectively guarantees that the request will not be processed until after the specified date and time. The request remains in the queue while the /AFTER switch is in effect and gains priority by virtue of its age. Refer to the section at the beginning of this chapter, "Selecting Files by Date and Time," for more information about the date and time format.

You can use a plus sign (+) to delay processing the request for a specific time period. For example, /AFTER=+3 delays processing for at least 3 hours.

/DENSITY=mode

Specifies the density to be used for the tape to be mounted (provided the tape unit supports that mode). Available modes are

800	(800 bits per inch)
1600	(1600 bits per inch)
6250	(6250 bits per inch)
ADM	(Automatic Density Matching — on reads)
LOW	
MED	(reverts to low on a dual-density unit)
HIGH	

If you intend to use a tape on another unit, be careful about choosing LOW, MEDIUM, or HIGH: "low" or "high" on one unit may not be compatible with the densities on another unit, which would prevent reading from the tape.

/DIRECTORY=pathname

Creates the link file in the specified directory. If you omit this switch, the system creates the link file in your initial directory.

If you use this switch, the corresponding DISMOUNT command should include /DIRECTORY=pathname as well.

/EXTEND

Extends the list of tape volume IDs (volids) if too few were specified (/VOLID switch). This switch is most useful when you are writing material to tape and you do not know how many volumes will be needed.

/HOLD

Holds the entry in the queue. You can later release the entry via the QUNHOLD command.

/IBM

Mounts a tape labeled in IBM format. The EXEC utility informs the system operator that the tape has an IBM–format label.

/NOPEND	Does not suspend your CLI until an operator responds. The system posts your request to the mount queue and displays the sequence number of the request. You can display its status by using the QDISPLAY command or cancel it by using the QCANCEL command with the request's sequence number.
/NOTIFY	Sends a message to your terminal upon completion of this request.
/QPRIORITY=n	Assigns priority n to the request. The highest priority is 1; the lowest is 255. You cannot assign a priority that is higher than the one specified in your user profile. If you omit this switch, the system assigns a priority based on the following formula (where m represents the priority specified in your user profile): $n = (m + 255) / 2$
/READONLY	Instructs the system operator to remove the write enable ring, and sets the ACL of any tape mounted from this command to username,RE.
/S	Stores the job sequence number in the current CLI String so that you can use the number as an argument to commands via the !STRING pseudomacro.
/STRING	(CLI32 only.) Same as /S.
/VOLID=volid	Restricts the mount request to labeled tape, to the tape volume with the specified volid (written by the LABEL utility). You can specify multiple volumes with multiple /VOLID switches. EXEC will tell the operator to change volumes as needed, and EXEC will check that each tape volid is mounted in the order specified here. For multiple volume requests, you can include all volid names, or you can use the /EXTEND switch, which allows additional volumes to be mounted when the ones you specify with the /VOLID switch have all been written to (or read from). After the operator types a command of the form `CONTROL @EXEC MOUNTED tape–unitname` the system creates a link file in your initial directory. This link file resolves to @LMT:first–volid.

MOUNT Example 1

```
) MOUNT/VOLID=SAL01/EXTEND  TFILE  Please mount scratch tape with ring. ⤶
```

This command requests the system operator to mount a tape that has a write–enable ring inserted. It specifies the linkname TFILE, which you will use later to refer to the tape. The system console displays messages such as these:

```
From PID 5: (XMNT)  29-Oct-1991 16:24:04
                    *******************
                    Labeled Mount Request
                    *******************
                    MID=45    ,            USER=SAL
                    USER PID=125           Requestor PID =142
                    Volumes:    SAL01

From PID 5:
                    Mount Volume           SAL01
                    Settings:              Default Density
                    User Message:          Please mount scratch tape with ring

                    Respond:               CX MOUNTED [@unitname]
                              or           CX REFUSED
```

After mounting the tape, the operator enters a MOUNTED command of the following form:

```
CONTROL @EXEC MOUNTED  tape–unitname
```

The CLI prompt appears at the SAL's terminal. He or she can now access the mounted tape by using the linkname TFILE. For example,

```
) DUMP_II/V :UDD:[!USERNAME]:TFILE:FILE1   # ⤶
```

MOUNT Example 2

```
) MOUNT TAPE1 Please mount tape number T15487 ⤶

) DUMP_II/V  TAPE1:3  +.SR
.
.(Program displays filenames dumped)
.
) REWIND TAPE1
) DISMOUNT TAPE1  Thanks — that's all until tomorrow. ⤶
```

The MOUNT command requests the system operator to mount a tape, which will be referred to by the linkname TAPE1. The DUMP_II utility dumps from the working directory to tape file 3 all files with the .SR suffix. The /V switch tells the program to display dumped filenames. The REWIND command returns the tape to its beginning. The DISMOUNT command requests the operator to remove the tape from the unit.

!OPERATOR *Pseudomacro*

Expands to ON or OFF, depending on whether the system operator is on or off duty.

Format

[!OPERATOR]

This pseudomacro returns either ON or OFF, according to whether or not the system operator is on duty. (The operator uses a CONTROL @EXEC command to indicate that he/she is on or off duty.)

NOTE: This pseudomacro has no relation to the CLI OPERATOR command, which pertains to the operator mode for working with labeled diskettes.

- No arguments.
- No macro name switches.
- Requirement: *Standard.*
- See also: DUMP, LOAD, MOUNT

Why Use It?

Use the !OPERATOR pseudomacro to determine whether or not the system operator is on duty. Certain CLI commands, such as MOUNT, require the system operator to perform an action for you. You can use !OPERATOR to check for the operator's presence before issuing such a command.

!OPERATOR Example

(within a macro)

```
[!equal, [!operator], on]
            qbatch  xeq  my_prog1  file2
[!else]
            write Operator off duty — try again later.
[!end]
```

This macro will execute program MY_PROG1 in batch if the system operator is on duty; otherwise it will write the message *Operator off duty — — try again later.*

PASSWORD *Command*

Sets a password for your CLI process (CLI32 only).

Format

```
PASSWORD
```

This command allows you to enter or change a password for CLI32. You will need to specify this password before the CLI will obey a LOCK or UNLOCK command.

An acceptable password can be 1 through 32 characters long. Choose any characters that are not control characters or function keys. The system converts any lowercase letters to uppercase ones.

Note that this password is specific to the CLI process you are running. It has no relation to the username/password you type to log on.

The CLI-specific password is discarded when you log off or this CLI terminates. However, if you write the password to disk with a command of the form PASSWORD/WRITE=path, you can re-establish that password later using a command of the form PASSWORD/READ=path. This eliminates the need to recreate the password. If you want, your log-on macro can include a command of the form PASSWORD/READ=path to automatically re-establish the password.

If you want to save your CLI password to a file, you must use the PASSWORD command with the /WRITE= switch, which creates the file and automatically encrypts the content. The PASSWORD command with the /READ= switch and the LOCK command with the /FILE= switch expect that the content of the named file is encrypted and read it in without change. The LOCK and UNLOCK commands encrypt the password you type, and compare it with the one read from the file. If the file that was read was not encrypted, the comparison fails.

If you lock your CLI by reading an unencrypted password, you cannot UNLOCK it. The BYE command will not work and you will have to ask the system manager or an operator with superprocess privilege to terminate your process. (See Example 4.)

PASSWORD (continued)

The /PROMPT or the /NOPROMPT switch used with the PASSWORD command determines whether you must enter the CLI password when using the LOCK command. The switch setting is specific to the CLI that is running, remaining in effect only as long as the process exists.

- No templates.
- No arguments.
- Requirement: *Standard.*
- See also: LOCK, UNLOCK.

Why Use It?

Use the PASSWORD command with CLI32 to prevent other people from malicious or unauthorized use of your CLI. Also, to safeguard the system console, system managers can use it to lock CLI32; this lets them avoid running LOCK_CLI.

NOTE: Use PASSWORD/READ= and LOCK/FILE= only to retrieve a password saved in a file created by PASSWORD/WRITE=.

Command Switches

The section "Universal CLI Switches," earlier in this chapter, describes switches /1, /2, /L, /L=pathname, and /Q, which you can use with all commands. That section also explains the CLI32 switches /STR= and /ESTR=.

/CHANGE	Starts the sequence of events necessary to change your CLI password. You must then give your old CLI password once and then the new CLI password twice.
/NOPROMPT	Does away with the prompt for and the need to enter the CLI password when using the LOCK command. The lock function is carried out automatically when the LOCK command is issued. This switch may be used when you first set up the CLI password or when you change it.
/PROMPT	Restores the prompt for and the need to enter the CLI password when using the LOCK command. The password must be entered and validated before the CLI lock function is carried out. This switch may be used when you first set up the CLI password or when you change it.
/READ=pathname	Read the CLI password from the specified file. You can use this switch to change your CLI password to one you previously saved with /WRITE=pathname.

PASSWORD (continued)

/WRITE=pathname	Write the current CLI password to the specified file. The system encrypts the password before placing it in the file and finishes by assigning the following ACL to the file. username,RE The file cannot exist before you issue PASSWORD with the /WRITE switch. If a file with that name does exist, either rename or delete it, or choose a different name for the new password file.

PASSWORD Examples

Typical dialog with the PASSWORD command follows. For security reasons, the CLI does not echo your password.

PASSWORD Example 1

The following example shows creation of a password.

```
) PASSWORD ↵
New CLI password: NPS ↵          (Password never echoes.)
Confirm password: NPS ↵
Password accepted.
```

PASSWORD Example 2

This example shows an attempt to change a password.

```
) PASSWORD/CHANGE ↵
Old CLI password: NPT ↵
Warning: Password did not match. Password not changed.
```

PASSWORD Example 3

This example places the existing password (encrypted) in a file named PW, where a later PASSWORD/READ=PW command can re–establish it.

```
) PASSWORD/WRITE=:UDD:[!USERNAME]:PW ↵
) BYE ↵
AOS/VS II CLI32 Terminating... ↵
```

(User logs on again.)

```
) PASSWORD/READ=PW ↵
```

PASSWORD Example 4

This example shows correct and incorrect command sequences for creating and saving a password. After following the correct sequence, the user can unlock the keyboard and exit the CLI. After following the incorrect sequence, the user types a password that the CLI encrypts and compares with the unencrypted password read from the file. Since the passwords do not match, the CLI remains locked.

Correct Procedure

```
) PASSWORD/WRITE=encrypted ↵
New CLI password: my_password ↵
Confirm password: my_password ↵
Password accepted.
) LOCK/FILE=encrypted ↵
) SUPERUSER ON ↵
) Error: Command is locked
) UNLOCK ↵
Password: my_password ↵

) BYE ↵
AOS/VS II CLI32   Terminating ...
```

Incorrect Procedure

```
) WRITE/L=unencrypted my_password ↵
) PASSWORD/READ=unencrypted ↵


) LOCK/FILE=unencrypted ↵
) SUPERUSER ON ↵
Error: Command is locked
) UNLOCK ↵
Password: my_password ↵
Warning: Password did not match
) BYE ↵
Error: Command is locked, BYE
```

PATHNAME *Command*

Displays a file pathname starting at the root directory.

Format

PATHNAME filename *[...]*[1]

[1] Multiple arguments available in CLI32 only.

This command returns a complete pathname for the specified file. The complete pathname begins at the system root directory.

NOTE: The CLI can return a full pathname for any file that resides in either your working directory or a directory on your search list. If the file is neither in the working directory nor in a directory on your search list, you must provide a pathname, not a filename; if you don't, the CLI will report *Warning: File does not exist*

- No templates.
- No argument switches.
- Requirement: *Standard.*
- See also: !PATHNAME, SEARCHLIST.

Why Use It?

Use the PATHNAME command to determine the full pathname of a file. For example, if a file exists within a directory on your search list, you can refer to the file by its name only, without giving a full pathname. Sometime you may need to know exactly where this file is located; for example, if you want to move a copy of it somewhere. The PATHNAME command tells you where the file is.

Command Switches

The section "Universal CLI Switches," earlier in this chapter, describes switches /1, /2, /L, /L=pathname, and /Q, which you can use with all commands. That section also explains the CLI32 switches /STR= and /ESTR=.

PATHNAME Example 1

```
) PATHNAME  FAS.CLI )
:UDD:TOM:MACROS:FAS.CLI
```

This command requests the complete pathname for the file FAS.CLI.

PREFIX

Command

Displays or sets the CLI prefix string.

Format

PREFIX *[argument] [...]*

This command either shows the current value of the CLI prefix string, or lets you define a new prefix. The CLI displays its prefix string (usually referred to as the CLI *prompt*) when it is ready to accept input.

The CLI prefix is normally a right parenthesis. You can change the prefix by supplying a string of up to 24 characters (for CLI16) or 80 (for CLI32). The CLI displays any prefix and follows it with a blank space.

With CLI32, the prefix is part of the environment; you can specify a different prefix for a different level. With CLI16, the prefix remains constant through all environment levels.

- No templates.
- No argument switches.
- Requirement: *Standard.*
- See also: PROMPT.

Why Use It?

PREFIX is the only way to modify the CLI prompt character [)]. A related command, PROMPT, lets you specify CLI *commands* for the CLI to execute before it displays the prompt character.

If you frequently log on to remote terminals, you could include a PREFIX command in your startup macro on each system to show that system's name as part of the CLI prefix. (See Example 2.)

Command Switches

The section "Universal CLI Switches," earlier in this chapter, describes switches /1, /2, /L, /L=pathname, and /Q, which you can use with all commands. That section also explains the CLI32 switches /STR= and /ESTR=.

/7BIT	(CLI32 only.) Tells the CLI to remove all parity bits on the output of the command. The CLI generates an error when it expands an !ASCII argument to a special character such as an angle or square bracket, or parenthesis. Add a high order (parity) bit to the !ASCII argument to prevent the CLI from interpreting it, and with this switch produce 7-bit output of the desired character. See the WRITE command in this chapter for a list of character codes.

For example, the ASCII code 074 expands to a left bracket in the following command, which produces an error:

```
) PREFIX [!ASCII 074]ABLE[!ASCII 076] ⤶
Error: Unmatched [ ( or <, expanding !ASCII 74.
```

Changing the ASCII codes to 274 and 276 and adding /7BIT to PREFIX prevents the error so that the desired prompt is created.

```
) PREFIX/7BIT [!ASCII 274]ABLE[!ASCII 276] ⤶
<ABLE>
```

/HISTORY=ON	(CLI32 only.) Inserts the number of the last command in this CLI's history buffer in the prefix display. The default is off. See the HISTORY command description.
/HISTORY=OFF	(CLI32 only.) Removes display of the history command count the from the prefix display. The default is off, so you only need to use this switch if /HISTORY=ON was used previously. See the HISTORY command description.
/I	Returns the CLI prompt to its default (initial) value — a right parenthesis. (No argument is allowed.)
/INITIAL	(CLI32 only). Same as /I.
/LEVEL=n	(CLI32 only.) Sets the prefix to the one used at the specified environment level (n). The integer n can be absolute or relative. An unsigned integer makes n absolute; for example, /LEVEL=2 means "use the value on level 2." A leading minus sign (−) makes n relative, n being the number of levels above the current level (toward 0). For example /LEVEL=−2 means two levels above the current one. We recommend /LEVEL over /PREVIOUS.
/P*[=n]*	Without *=n*, sets the prefix to the one used in the previous CLI environment. (Omit the pathname argument.) With *=n* (CLI32 only), sets the prefix to the one used in the specified environment level. The *n* specifies the number of levels above the current level (toward 0).
/PREVIOUS*[=n]*	(CLI32 only.) Same as /P.

PREFIX Example 1

You frequently use the network to log on a remote terminal on a system named CYRANO. To help you keep track of where you are in the network, you include the following command in your startup macro on CYRANO.

```
prefix CYRANO[!ASCII 251]
```

The pseudomacro [!ASCII 251] expands to a right parenthesis. When you log on CYRANO, your CLI prompt will appear as

CYRANO)

To sets the CLI prefix to its initial value, a right parenthesis, use the /I switch.

```
CYRANO) PREFIX/I ⤶
)
```

PRIVILEGE *Command*

Sets or displays the privilege settings.

Format

```
PRIVILEGE [ SUPERPROCESS   ] [ ON  ]
          [ SUPERUSER      ] [ OFF ]
          [ SYSTEMMANAGER  ]
```

This command allows you to set or display the current privileges.

If you omit arguments, the CLI displays your current privilege modes. If you turned on Superuser and/or Superprocess via those commands, the display indicates this.

If you supply one argument, which must be SUPERPROCESS, SUPERUSER, or SYSTEMMANAGER, the CLI displays OFF or ON. If you supply two arguments, which must be SUPERPROCESS, SUPERUSER, or SYSTEMMANAGER, and ON or OFF, and your profile grants the privilege, the CLI changes the privilege mode.

The privilege modes are part of the CLI environment level. You can set modes at one level, push to another level, and change a mode. When you use POP to return to the previous level, the modes set at that level will be restored.

The prompts corresponding to each combination of privileges follow.

Prompt	**Privileges**
)	None
Sm)	System Manager
Sp)	Superprocess
Su)	Superuser
SmSp)	System Manager and Superprocess
SmSu)	System Manager and Superuser
SpSu)	Superprocess and Superuser
SmSpSu)	System Manager, Superprocess, and Superuser

NOTE: Use any of these privileges with caution. They override certain safeguards of the operating system and their improper use could severely harm another user.

If you change the prefix (for example, to add a network hostname to it), and you have one or more privileges set on, the CLI inserts an exclamation point between the privilege prompt(s) and your prefix. For example, if you change the prefix to VENUS) and have Superuser on, the displayed prompt will be

Su!VENUS)

PRIVILEGE (continued)

- No templates.
- No argument switches.
- Requirement: *Standard* to display your privilege modes, Systemmanager to set Systemmanager privilege on, Superprocess to set Superprocess privilege on, and Superuser to set Superuser privilege on.
- See also: SUPERPROCESS, SUPERUSER.

Why Use It?

Use the PRIVILEGE command to turn System Manager mode on, perhaps to change the system date or time. You can also use it to control Superprocess or Superuser mode, although there are other commands (SUPERPROCESS and SUPERUSER) that do the same thing. For example, you need Superprocess on to terminate another user's process if he or she cannot get the CTRL−C CTRL−B sequence to respond properly.

Command Switches

The section "Universal CLI Switches," earlier in this chapter, describes switches /1, /2, /L, /L=pathname, /Q, /STR=, and /ESTR=, which you can use with all CLI32 commands.

/KILL	Turns all privileges off.
/LEVEL=n	(CLI32 only.) Sets the current privileges to those of the specified environment level (n). The integer n can be absolute or relative. An unsigned integer makes n absolute; for example, /LEVEL=2 means "use the value on level 2." A leading minus sign (−) makes n relative, nn being the number of levels above the current level (toward 0). For example /LEVEL=−2 means two levels above the current one. We recommend /LEVEL over /PREVIOUS.
/P*[=n]*	Without *=n*, sets the privileges to those in the previous CLI environment level. With *=n* (CLI32 only), sets the privileges to those in the specified environment level. The *n* specifies the number of levels above the current level (toward 0).
/PREVIOUS*[=n]*	(CLI32 only.) Same as /P.

PUSH Example 1

```
) LEVEL ⤶
Level 0
) PUSH/V ⤶
Level 1
```

The first command displays the current environment level. The PUSH command moves down to the next level, and verifies the move.

PUSH Example 2

```
) SEARCHLIST ⤶
:UTIL
) PUSH ⤶
) SEARCHLIST :UDD:COMMON:SPECIAL_UTILITIES [!SEARCHLIST] ⤶
) XEQ  MYPROG ⤶
) POP ⤶

) SEARCHLIST ⤶
:UTIL
```

This user wants to run program MYPROG in directory SPECIAL_UTILITIES. The SEARCHLIST command displays the current level search list, :UTIL.The PUSH command changes the current environment level. The next commands change the current search list, and then execute a program. Finally, the POP command returns to the original environment level, restoring the previous search list.

PUSH Example 3

The following example is a macro version of the previous one. It begins with PUSH and PROMPT POP commands to ensure that the original environment will be restored if the macro is interrupted.

```
PUSH; PROMPT POP
SEARCHLIST :UDD:COMMON:SPECIAL_UTILITIES [!SEARCHLIST]
XEQ  MYPROG
POP
```

QBATCH

Command

Creates and submits a job for batch processing.

Format

QBATCH cli_command_line *[argument] [...]*

This command creates a batch job to execute the CLI command or macro in the specified command line and submits it to the batch queue for processing. *Batch processing* is an alternative to working interactively. In batch mode, the system executes a *job* consisting of one or more CLI commands. The job is placed in a batch queue, which the system processes on its own time. Your terminal remains free for other activity.

To run the batch job, the system creates a batch input file containing CLI commands to set the working directory, search list, and default ACL to the settings they had when you issue the command. The batch input filename has the form ?.pid.CLI.n.JOB and it is created in the working directory. After the job runs, the system deletes the batch input file, unless an error prevents the job from completing. You can delete obsolete batch input files if you want.

You can specify a queue other than the default batch input queue with the /QUEUE=queuename switch.

You can check the status of batch jobs with the QDISPLAY command.

- Accepts any argument switches appropriate for the specified job.
- Requirement: *Standard*.
- See also: !LOGON, QDISPLAY, QCANCEL, QHOLD, QMODIFY, QSUBMIT, QUNHOLD.

Why Use It?

Use the QBATCH command to run one CLI command or macro in batch mode. Using batch mode allows you to submit a job for processing without waiting for the system to act on your request. You can use the batch queue to run a job overnight when the system may not be as heavily used.

Batch processing is also useful for jobs that contain several steps, such as compiling and linking code to build a program file. You can submit this type of a job to the batch queue so that you do not have to monitor its progress.

QBATCH allows only one command line. If the job you want to run requires more than one CLI command line, you can either combine the commands in a macro, or use the QSUBMIT command.

QBATCH Command Switches

The section "Universal CLI Switches," earlier in this chapter, describes switches /1, /2, /L, /L=pathname, and /Q, which you can use with all commands. That section also explains the CLI32 switches /STR= and /ESTR=.

/AFTER=date:time	Processes this request after the specified date and time. This differs from the /AFTER switch used with a "last" switch like /TLM. Use the format dd–mon–yy for date; use hh:mm:ss for time; if you specify both, separate them with a colon. Minutes and seconds are optional. For example, the switch /AFTER=12–FEB–91:14 specifies after February 12, 1991, 2:00 p.m. Or you use the form +hh:mm:ss alone to specify a delay from the current time. For example, /AFTER=+3 means "Execute the command as soon as possible, 3 hours from now."
/CPU=time	Limits CPU time for batch jobs to the specified amount of time. Use the format hh:mm:ss to specify the time (minutes and seconds are optional). You must allow enough time for all processes created in the batch job. This switch applies only if the operator has set a time limit for jobs in the stream; otherwise the switch is ignored. If a time limit is in effect and you specify a limit that exceeds that value, the system does not process your request.
/DESTINATION=string	Prints the specified string in block letters at the top of any header or trailer pages. If you omit this switch, the current username is printed.
/HOLD	Holds this job in the queue. You can later use the QUNHOLD command to release the job.
/I	Takes the contents of the file from subsequent lines of the @INPUT file. The CLI will ignore pseudomacros as input. You must terminate the input with a line containing a single right parenthesis,), and a NEW LINE (no arguments allowed).
/INPUT	(CLI32 only.) Same as /I.
/JOBNAME=name	Assigns the job this name, which you can use later with a QHOLD, QUNHOLD, or QCANCEL command. The name must contain at least one alphabetic character. If you omit this switch or do not specify a name, the system assigns no name (null) for the entry.
/M	Takes the contents of the file from subsequent lines of the current macro body; the CLI won't interpret pseudomacros in these lines. The last line of the macro file must contain a single right parenthesis,). (No arguments are allowed.)
/MACRO	(CLI32 only.) Same as /M.

/NORESTART	If the system fails while processing this job, this switch does not restart it when the system comes up. By default, the system restarts the job.
/NOTIFY	Sends a message to your terminal when this job is completed. By default, no message is sent.
/OPERATOR	Runs this job only if a system operator is on duty. You should use this switch if the batch job contains a MOUNT request.
/QLIST=pathname	Causes the CLI @LIST output to be placed in the specified file. If you do not use this switch, @LIST output will be placed in a temporary file (:QUEUE:user.LIST.sequence–number) and enqueued to the batch queue's list queue.
/QOUTPUT=pathname	Causes the batch job output to be placed in the specified file. If you do not use this switch, output will be placed in a temporary file (:QUEUE:user.OUT.sequence–number) and enqueued to the batch queue's output queue.
/QPRIORITY=n	Assigns priority n to this job. The highest priority is 1; the lowest is 255. You cannot assign a priority that is higher than the maximum queue priority specified in your user profile. If you omit this switch, the system assigns a priority based on this formula, where m represents the maximum queue priority specified in your user profile: n = (m + 255) / 2
/QUEUE=queuename	Submits the job to the specified queue, instead of to the default batch queue.
/S	Stores the job sequence number in the current CLI String so that you can use the number as an argument to commands via the !STRING pseudomacro.
/STRING	(CLI32 only.) Same as /S.
/V	Displays the name of the batch job file.
/VERIFY	(CLI32 only.) Same as /V.

/QPRIORITY=n	Assigns priority n to this job. The highest priority is 1; the lowest is 255. You cannot assign a priority that is higher than the maximum queue priority specified in your user profile. If you omit this switch, the system assigns a priority based on this formula where m represents the maximum queue priority specified in your user profile: n = (m + 255) / 2
/QUEUE=queuename	Submits the job to the specified queue rather than to the default queue (LPT). The queue type must be PRINT.
/S	Stores the job sequence number in the current CLI String so that you can use the number as an argument to commands via the !STRING pseudomacro.
/STRING	(CLI32 only.) Same as /S.
/SORT	(CLI32 only.) Sorts alphabetically the filenames this command processes. To see the names, you must also use the /V switch.
/TITLES	Prints a title line at the beginning of each page. The title line includes the file pathname, date, time, and page number. If you omit this switch, the system prints no titles. Title lines do not print if the command included either the /BINARY or the /PASSTHRU switch.

/TRAVERSE=directory–type

(CLI32 only.) Specifies directory types to traverse (go through) while executing this command. Table 2–8 contains valid values of directory type. You can use this switch to include directory types, such as /TRAVERSE=CPD, and to exclude directory types, such as /TRAVERSE=\CPD. Numbers from Table 2–8 are also valid values of directory type, such as /TRAVERSE=10–11. Without this switch, a command such as

```
QPRINT/TYPE=\CPD #:PROJ.–
```

will apply to *all* directories even though /TYPE=\CPD is in the command. With this switch, commands such as

```
QPRINT/TRAVERSE=\CPD #:PROJ.–
```

give expected results.

/TYPE=typecode	(CLI32 only.) Specifies one or more types of files to process. Valid values of type codes are in Table 2–8. Some pertinent ones are TXT, UDF, and UNX.
/V	Displays the names of the queued files. This switch is helpful if you used a template or date/time switch to specify files.
/VERIFY	(CLI32 only.) Same as /V.

QPRINT Example 1

```
) QPRINT FILE1 FILE2 ⤶
Queued, Sequence number = 655, Qpriority = 127
Queued, Sequence number = 656, Qpriority = 127
```

This command sends FILE1 and FILE2 to the line printer output queue.

QPRINT Example 2

```
) QPRINT/QUEUE=LASER  MY_REPORT ⤶
Queued, Sequence number = 657, Qpriority = 127

) QDISPLAY/QUEUE=LASER ⤶

LPT   PRINT  Open
* 657 D    JONES    :UDD1:JONES:FILE3

Flags explanation:
* = Active
```

This command prints MY_REPORT on the laser printer, queuename LASER. The QDISPLAY command shows that the job is active in its queue.

QPRINT Example 3

```
) QPRINT/COPIES=3/PAGES=75/TITLES  FILE4 ⤶
Queued, Sequence number = 658, Qpriority = 127
```

This command prints three copies of FILE4 and includes titles on each page. If the system operator has limited printing in this queue to less than 75 pages, the job will not run but will wait in its queue.

QPRINT Example 4

```
) QPRINT/QUEUE=TZONE:LASER2  :NET:MSIS:UDD:CHRIS:FINAL_SPEC ⤶
```

This command submits the file FINAL_SPEC (located in the directory :UDD:CHRIS directory on host MSIS) to a queue called LASER2 on the remote host TZONE. The full network pathname is needed because the file is not on the same host as the remote queue.

QPRINT Example 5

```
) QPRINT/COPIES=5/AFTER=20-APR-90:18:30  SET_IX ⤶
```

This command prints five copies of file SET_IX after 6:30 p.m. on April 20, 1990.

!READ *Pseudomacro*

Displays a text string, and expands to the typed response.

Format

```
[!READ { text                      } ]
       { /FILEID=file-ID text [...]¹ }
```

¹ Multiple arguments available in CLI32 only.

The first form of this pseudomacro displays the specified text on @OUTPUT (such as a video display screen), and then expands to the character(s) received from @INPUT (usually the keyboard). Consequently, you use this pseudomacro to make a macro interactive. In such instances, the CLI displays text as a prompt, and pauses in its execution until you make an entry on your terminal keyboard. The !READ pseudomacro then expands to the value of your keyboard input.

The second form of the command can make reads from a file interactive. You identify the file by file–ID.

- No templates.
- No argument switches.
- Accepts a macro name switch (described later).
- Requirement: *Standard.*
- See also: STRING, VARn, !EQUAL, !NEQUAL.

Why Use It?

Use the !READ pseudomacro to prompt a macro user for input, and then use that input in a command. Also, see the description of the WRITE command for an explanation of how the !READ pseudomacro can get a response from a menu of choices on a terminal's screen.

Macro Name Switches

/EOF=string	Specifies the string the system will return when it encounters the end of the file. This value of string overrides any value you specified for the OPEN or READ command /EOF=string switch.
/FILEID=file–ID	(CLI32 only.) Identifies the file you want to read. The default file ID is the file's name (not pathname), without any trailing suffix. You can learn the file IDs of all open files by typing the OPEN command without an argument.

!READ (continued)

/LENGTH=n	(CLI32 only.) Tells the CLI to end the read when n characters have been typed. A terminating New Line character is not necessary.
/S	Discards all the text typed in after a semicolon or left bracket ([). If you omit this switch, any text following a semicolon must be a valid CLI command or macro call; any text following a left bracket must be a valid pseudomacro, program, or file name, followed by a right bracket; otherwise an error will occur.
/SECURE	(CLI32 only.) Same as /S. Not valid together with the /FILEID switch.

!READ Example 1

(within a macro)

```
string [!read Do you want to continue?,,]
[!equal, [!string], Y]
            comment Continue processing here.
            ...
[!else]
            write Done!
[!end]
```

This macro displays the prompt *Do you want to continue?* and then accepts a response from @INPUT (normally the keyboard), placing the response in the CLI String. The second statement uses the contents of the CLI String to determine whether or not to execute the group of statements that follow.

!READ Example 2

(within a macro)

```
...
xeq  masm  [!read What file do you want to assemble?,,]
...
...
```

This command displays the prompt

*What file do you want to assemble?*VV

(that includes two trailing spaces, shown as VV) and uses the response as the argument to the XEQ MASM statement. The specified filename effectively replaces the pseudomacro in the command line.

!READ Example 3

(within a macro)

```
...
string [!read/s Enter your comment:,,]
...
...
```

Any semicolon typed in response to the following prompt will end the input.

*Enter your comment:*VV

Any characters that follow the semicolon are ignored, but no error message appears. If the switch /S wasn't present and the string typed as a response to the prompt contained a semicolon, the system would try to execute the characters after the semicolon as a CLI command. An error message would appear if the characters don't form a valid CLI command.

!READ Example 4

Suppose that you want to write a macro that prompts you to choose a program to run (either PREP.PR or CAL.PR), pauses while you make your selection by entering P for PREP.PR or anything else for CAL.PR, and then runs the specified program.

The following macro performs the above steps.

```
STRING [!READ Type P to run PREP.PR or anything else to run CAL.PR:,]
[!EQUAL P,[!STRING]]
            XEQ  PREP.PR
[!ELSE]
            XEQ CAL.PR
[!END]
```

When you run this macro, the first line displays the prompt

Type P to run PREP.PR or anything else to run CAL.PR:

followed by a space. After you type a response and press NEW LINE, or just press NEW LINE, the macro assigns your input to the String. The second line of this macro compares the String to the character P. If the String equals the character P, the macro executes PREP.PR. Otherwise, the macro executes CAL.PR.

!READ Example 5

(within a macro)

```
...
[!READ/LENGTH=1  Please answer Y or N:,,]
...
```

A one-character response without a New Line delimiter executes the read.

!READ Example 6

The following macro called FILE_READ.CLI contains a loop that reads a line from a file and writes it to the screen. When the string specified by /EOF= is recognized, loop execution terminates.

```
\\ This macro reads and lists the input file line by line.
OPEN/READ MESSAGES                          \\ Opens the input file
   [!LOOPSTART]
   STRING [!READ/FILEID=MESSAGES/EOF=ZZZZZZ]    \\ Reads a line from file
      [!EQUAL,([!STRING]),(ZZZZZZ)]             \\ Tests for EOF string
         [!EXIT/LOOP]                       \\ Skips to next command after loop
      [!ELSE]
         WRITE [!STRING]                    \\ Writes line from file to screen
      [!END]
   [!LOOPEND]
CLOSE/FILEID=MESSAGES                       \\ Closes the input file
```

Create a text file as input for the macro:

```
) CREATE/I MESSAGES ⤶
)) Now is the time for all good people to help save the environment. ⤶
)) By participating in our recycling campaigns you promote this goal. ⤶
))) ⤶
```

Now execute the macro:

```
) FILE_READ ⤶
MESSAGES
Now is the time for all good people to help save the environment.
By participating in our recycling campaigns you promote this goal.
```

To verify that end of file was found, you can write the contents of STRING after the macro finishes.

```
) WRITE [!STRING] ⤶
ZZZZZZ
```

For the fourth line of the READ_FILE.CLI macro, substitute the following, which adds the switch /LENGTH=40.

```
STRING [!READ/FILEID=MESSAGES/LENGTH=40/EOF=ZZZZZZ]
```

Reading the same file, the macro now limits each line to 40 characters.

```
) FILE_READ ⤶
MESSAGES
Now is the time for all good people to h
elp save the environment.
By participating in our recycling campai
gns you promote this goal.
```

Removes a logical disk unit (LDU) from the working directory.

Format

```
RELEASE ldu–pathname [...]
```

This command releases an initialized logical disk unit (LDU, which is a directory made up of one or more physical disks) from the directory in which it was initialized.

The INITIALIZE command adds an LDU to the working directory. After executing the INITIALIZE command, the system displays the LDU filename. Use that filename to release the LDU. If you do not remember the LDU filename, you can display it with a command of the form

```
FILESTATUS/TYPE=LDU  pathname–of–directory–where–ldu–was–initialized:+ ↵
```

Usually, LDUs are initialized in the root directory.

The system will not release an LDU (but will display a *Directory in use* error message) if any user's working directory is in that LDU or if the LDU is on any user's search list. If you receive this error message, you can broadcast a message advising users to log off or change their search lists. With AOS/VS II, you can force release with the /FORCE switch. Shutting down the operating system releases all initialized LDUs.

- No templates.
- No argument switches.
- Requirement: Execute and Write access to the directory in which the LDU was initialized and Owner access to the LDU; otherwise, Superuser on.
- See also: INITIALIZE (to add an LDU to the working directory), MIRROR (to start synchronizing another image of an LDU).

Why Use It?

Use the RELEASE command to remove a logical disk unit (LDU) that was added to the working directory via an INITIALIZE command. You may want to do this to check the disk unit and/or do preventive maintenance.

At shutdown, usually the system DOWN.CLI macro releases the LDUs individually; if not, system shutdown releases them.

RELEASE Command Switches

The section "Universal CLI Switches," earlier in this chapter, describes switches /1, /2, /L, /L=pathname, and /Q, which you can use with all commands. That section also explains the CLI32 switches /STR= and /ESTR=.

/FORCE	(AOS/VS II only) Forcibly releases the specified LDU(s). Closes all files on that LDU and any subordinate LDUs initialized in it, and aborts mirror synchronization, if in progress. A user who has files open on the LDU will lose any changes that have not been written to disk. If you attempt to force release an LDU that contains the ROOT, PAGE, or SWAP directories, you will receive an error message. If you want to release these system directories, you must first shut down the operating system. Use your system DOWN.CLI macro or begin ESD if you really do want to release the root LDU. After a forced release, any process that tries to access the LDU will receive the error message *Directory not available because the LDU was force released.* If the searchlist of any process includes a directory on the released LDU, and that process tries to use the search list, it receives the same error message; the process cannot execute programs or check its search list. To recover from these error conditions, the process must change its searchlist to omit any directory on the released LDU. If the initial working directory of a process is forcibly released, the process should change to the root directory and then to another valid directory.

RELEASE Example 1

Su) INITIALIZE @DPJ0 @DPJ1 ⤶	(Initializes the LDU of units DPJ0 and DPJ1.)
DATABASE	(System displays LDU filename.)
Su) DIRECTORY DATABASE ⤶	(Makes the LDU the working directory.)
Su) FILES/AS/S ⤶	(Lists files in the LDU primary directory.)
...	(System displays filenames.)
Su) DIRECTORY ^ ⤶	(Changes to previous directory, above the LDU; in preparation for releasing the LDU.)
Su) RELEASE DATABASE ⤶	(Releases the LDU.)

RELEASE Example 2

```
) SUPERUSER ON ↵                      (Turns Superuser on.)
Su) DIRECTORY : ↵                     (Makes the root the working directory. )
Su) INITIALIZE/S @DPJ1 ↵              (Initializes the disk into the current directory,
                                      the root, and puts the LDU name into the
                                      CLI string.)
Su) ACL [!STRING] +,RE ↵              (Sets the LDU ACL to Read and
                                      Execute for everyone.)
Su) WRITE/L=LDU_NAME [!STRING] ↵      (Writes the LDU's name into the file
                                      LDU_NAME from the CLI string.)
...                                   (Executes various CLI commands.)
Su) RELEASE [LDU_NAME] ↵              (Releases the LDU named in LDU_NAME.)
Su) SUPERUSER OFF ↵                   (Turns Superuser off.)
)
```

RENAME *Command*

Changes the name or pathname of a file.

Format

$$\text{RENAME pathname} \left\{ \begin{array}{l} \text{new–filename} \\ \text{full–pathname}^{1} \end{array} \right\}$$

[1] AOS/VS II only.

This command renames the file specified in pathname to new–filename. The pathname can contain the prefixes ^, =, :, and @. Use the first form of the command to rename a file within its current directory.

With AOS/VS II, you can relocate files specified in pathname to a different directory specified in full–pathname. This transfers the file from one directory to another (unlike the MOVE and COPY commands, which copy a file from one directory to another). The file you rename can be a directory. You must specify a full pathname, from the root (:), to the destination file. The destination filename cannot already exist in the destination directory. However, all directories above the destination file *must* exist; the RENAME command will not create directories or files. The destination pathname must be on the same LDU as the source pathname.

As with many other commands, you can use the !FILENAMES pseudomacro to specify multiple files. Include the !FILENAMES /NOEQUALS switch (CLI32 only) to avoid *Illegal filename* errors. You can also use other !FILENAMES switches. See the fourth and last examples below.

- No templates.
- No argument switches.
- Requirement: *Standard*.
- See also: MOVE, COPY, LOAD/LOAD_II, DUMP/DUMP_II, CREATE.

Why Use It?

Use the RENAME command to change the name of a file – perhaps to assign a shorter or more descriptive name to a file, or to correct a typing error. The pathname can contain a prefix like ^ (caret) or : (colon); without AOS/VS II, the new name cannot include a directory name.

On an AOS/VS II system, you can use the Rename Across Directories function (second form of the command) to relocate files to another directory. If you want only one copy of a file, this is a faster and more convenient method than using MOVE or COPY. See the sample macro (later in this section) for an example of how to transfer more than one file.

SCOM

Utility

Compares two ASCII text files.

Format

```
XEQ SCOM textfile1 textfile2
```

This program scans the specified files, comparing them line by line. If the files differ, the utility reports the first line where the files do not match. You can use the /L= switch (described later) to have SCOM list all the differences between the two files.

By default, SCOM uses a *match size* of four. This means that the utility looks for four consecutive matching lines. If only three out of four lines match, SCOM reports the entire group a mismatch. You can use the /MS= switch to change the utility's match size.

SCOM displays a header for each section indicating its location in the file in the form:

m/n (p)

where m is the page number, n is line number relative to the top of the page, and p is the absolute line number (counted from the start of the file). The description of the /V (verbose) switch shows the expanded header option.

The maximum number of characters per line for either file is 133, including the NEW LINE terminator. The maximum number of lines on a page for either file is 32767. (A line is any string that ends with an ASCII NEW LINE character.)

The maximum number of pages for either file is also 32767, where a page is any string ending in an ASCII form feed.

- Does not accept templates.
- Accepts utility switches (described later).
- Requirement: *Standard* (Execute access to the program file in :UTIL).
- See also: TYPE, QPRINT, DISPLAY, FILCOM.

Why Use It?

Use this utility to ensure that two text files are identical, or to discover how they differ.

SCOM works only with text files (data-sensitive records). To compare nontext files such as binary files or user data areas (UDAs), use the FILCOM utility.

SCOM Switches

/BEGIN=n	Sets the column number to start comparing lines. The default is 1.
/DNP	Display nonprintable characters. Well-known nonprintable characters display mnemonically; e.g., <NL>, <TAB>, and <FF>. Less common nonprintable characters display as octal values.
/END=n	Sets the column number to stop comparing lines. The default is 256 or the end of the line. The /END value must be equal to or greater than the /BEGIN value.
/EOL	Includes end-of-line characters and blank lines in the comparison. If you omit this switch, SCOM ignores EOL characters and blank lines.
/IGNORE	Excludes tabs, blanks and case sensitivity from the comparison (equivalent to using both /IGNORE=CASE and /IGNORE=WHITE).
/IGNORE=CASE	Excludes case sensitivity from the comparison.
/IGNORE=WHITE	Excludes tabs and blanks from the comparison.
/L	Writes the list of differences to the current @LIST file.
/L=pathname	Lists the text differences in the specified file.
/MAXLEN=n	Sets the maximum number of characters per line. The default is 256 characters per line.

SCOM (continued)

/MS=n	Sets the match size to n. (If you omit this switch, the default match size is 4.)
/NL	Suppresses list display. If you omit this switch, SCOM displays the mismatched portions of text from both files to @CONSOLE. Use the /L switch instead to redirect the listing to a file you can examine later.
/N	Expands the "m/n (p)" header to "Page m, Line n (Line p)."

SCOM Example 1

```
) XEQ SCOM/NL MYFILE YOURFILE )
```

Files differ starting on 1/4 (4)

This command compares the contents of the files MYFILE and YOURFILE.

SCOM Example 2

```
) XEQ SCOM/V/DNP/EOL/L=MYPROG.DIFS MYPROG.BU MYPROG ⤶

Files differ starting on Page 1, Line 9 (Line 9)

) TYPE MYPROG.DIFS ⤶

FILE MYPROG.BU, Page 1, Line 8 (Line 8) to Page 1, Line 12 (Line 12)

<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>

    integer OUTFILE, TOC<NL>
    parameter ( MAX_CPL = 86 ) <TAB>  ! Max number of chars per line.<NL>
    parameter ( MAX_PATHNAME = 54 )  ! Max number of chars in pathname.<NL>
    ; line 11<NL>
    ; line 12<NL>

<><><><><><><><><><><><><><><><><><><><>

FILE MYPROG, Page 1, Line 8 (Line 8) to Page 2, Line 3 (Line 12)

<><><><><><><><><><><><><><><><><><><><>

    integer OUTFILE, TOC<NL>
    <FF>
    parameter ( MAX_CPL = 136 ) <TAB>! Max number of chars per line.<NL>
    parameter ( MAX_PATHNAME = 54 )  ! Max number of chars in pathname.<NL>
    ; line 12<NL>

<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
```

The first command calls the SCOM utility to compare the files MYPROG.BU and MYPROG, writing the differences to MYPROG.DIFS. The TYPE command displays the contents of the file MYPROG.DIFS. The SCOM report shows the mismatched group of lines from each file.

SEND Restrictions

If you want to use the following characters within your message text, you must use the !ASCII pseudomacro with the appropriate octal value; otherwise the CLI will interpret the characters as syntax elements rather than literals:

( ) parentheses	< > angle brackets	
[ ] square brackets	, comma	; semicolon

Command Switches

The section "Universal CLI Switches," earlier in this chapter, describes switches /1, /2, /L, /L=pathname, and /Q, which you can use with all commands. That section also explains the CLI32 switches /STR= and /ESTR=.

/7BIT
: (CLI32 only.) Tells the CLI to remove all parity bits on the output of the command. The CLI generates an error when it expands an !ASCII argument to a special character such as an angle or square bracket, or parenthesis. Add a high order (parity) bit to the !ASCII argument to prevent the CLI from interpreting it, and with this switch produce 7-bit output of the desired character. (See the WRITE command in this chapter for a list of character codes.) For example, the following command produces an error:

```
) SEND 100 [!ASCII 050]Kyle[!ASCII 051] Meeting at 2 pm. ↵
Error: Unmatched [ ( or <, expanding !ASCII 50.
```

Changing the ASCII codes to 250 and 251 and adding /7BIT to SEND prevents the error so that the desired message is sent.

```
) SEND/7BIT 100 [!ASCII 250]Kyle[!ASCII 251] Meeting at 2 pm. ↵
```

/I
: Sends each input line that follows as a separate message. Subsequent lines may not contain conditionals; the CLI will not interpret pseudomacros in the context of /I. To end input, enter a line containing a single right parenthesis,). Use this technique to send a multiple-line message.

/INPUT
: (CLI32 only.) Same as /I.

/M
: Sends each subsequent line of the current macro file as a separate message. Macro lines may not contain conditionals; the CLI will not interpret pseudomacros in the context of /M. The macro sequence must end with a line that contains a single right parenthesis,). Use this technique to build the SEND command into a macro.

/MACRO
: (CLI32 only.) Same as /M.

SEND Example 1

```
) WHOS↵
...(System lists processes)...
00012 CHRIS ... CLI32.PR
) SEND 12 Are you ready to go to lunch?↵
```

SEND Example 1 (continued)

The WHOS macro supplied with the system lists all processes on the system. The message *Are you ready to go to lunch?* appears on the terminal of the person running PID 12.

SEND Example 2

```
) SEND/I 24↵
)) WE ARE MEETING TODAY AT 2.↵
)) CAN YOU MAKE IT?  – KIM↵
)) )↵
```

The first command uses the /I switch to take message input from the terminal. This allows you to send a message of more than one line. (The CLI executes a separate SEND command for each line.)

SEND Example 3

The next lines are part of a macro.

```
SEND/M  %2%
Time for the weekly meeting.
)
```

SEND Example 4

This example illustrates two users; one has the process running at console 46, the other at console 62. They use the SEND command for a brief dialague.

```
) SEND  @CON46 [!ASCII 207]  WAKE UP!↵
)
 From Pid 00062: WAKE UP!
```

This command sends a message to the process running at console 46. The message begins with an ASCII character that causes the receiving terminal to beep as the message is displayed.

The user at console 46 replies:

```
) SEND   @con62 What time is it?↵
```

The user at console 62 answers back:

```
)
 From Pid 00062: What time is it?
) SEND/7BIT   @con46 You can type WRITE [!ASCII 333]!TIME[!ASCII 335] to find out!↵
)
```

The message includes the ASCII characters for right and left square brackets, not possible without the /7BIT switch. It appears to the user at console 66 as follows:

```
)
From Pid 00046: You can type WRITE [!TIME] to find out!
```

SYSINFO *Command*

Displays information about the current system.

Format

SYSINFO

This command displays the following information about the current system environment:

- The operating system implementation (AOS/VS or AOS/VS II)
- The operating system revision number
- The microcode revision number
- The number of 2-Kbyte memory pages
- The filename of the master logical disk unit (LDU)
- The system identifier (as set via SYSID or defaulted)
- The filename of the booted operating system

SYSID command summary information:

- No arguments.
- Requirement: *Standard.*
- See also: SYSID, HOST.

Why Use It?

Use the SYSINFO command to display general information about your system. For example, you might want to know the operating system revision, master LDU filename, or the amount of memory. If you are using a virtual terminal over a network, you can use this command to check the system you are logged onto.

Command Switches

The section "Universal CLI Switches," earlier in this chapter, describes switches /1, /2, /L, /L=pathname, and /Q, which you can use with all commands. That section also explains the CLI32 switches /STR= and /ESTR=.

SYSINFO Example

This command displays information for the current system.

```
) SYSINFO ↵
System Implementation: AOS/VS
System revision: 07.69
Microcode revision: 8
System memory: 8192 Pages
Master logical disk: ROOT
Sysid: DANTE
Current system: :SYSGEN:DANTE.PR
```

SYSLOG *Command*

Starts, stops, or displays the status of system, Superuser, or CON0 logging.

Format

SYSLOG*[/switches] [filename]*

The SYSLOG command lets a system operator display the status of system logging (either on or off). The CLI32 /VERBOSE switch displays more information about SYSLOG logging options, Superuser logging, and CON0 logging. Other switches turn system logging, Superuser logging, or CON0 logging on or off. The filename argument lets the operator rename the system log file (:SYSLOG) and start logging in a new, empty :SYSLOG file, or, when used with the switches /CON0/START, to rename the CON0 log and start logging in a new, empty :CON0_LOG file.

System logging records details for each user like time logged on and off, devices used, and CPU time used.

Superuser logging logs events, caused by a superuser, that would normally be logged only if you chose full-detail logging. With CLI32, you turn on Superuser logging with the switches /SUPERUSER/START. You can write a program that uses an exclusion bit map to mask specific events for all users and/or an exclusion bit map to exclude specific events caused by a superuser.

CLI32 also lets you use CON0 logging to send to the file :CON0_LOG the I/O that appears on the system console, CON0.

When logging is on, the system always writes the log information to a file with the pathname :SYSLOG. The system uses a separate file, :ERROR_LOG, for error logging. The error log file contains all peripheral device error information. Logging is always on to the error log file.

The system and error log files are not directly readable. To read these files, or produce readable disk file reports from them, use the REPORT utility (described in *Managing AOS/VS and AOS/VS II*). The file CON0_LOG is readable: you can use the CLI TYPE command, described later in this chapter, or the BROWSE utility, described earlier in this chapter, to view it or to search for text strings in it.

- No templates or argument switches.
- Requirement: *PID2* or *System Manager privilege turned on.*
- See also: LOGEVENT, REPORT (in *Managing AOS/VS and AOS/VS II*).

Why Use It?

System log information is helpful when you want to monitor system use. Use the REPORT utility to obtain a printed report about the contents of a log file. Superuser logging (logged to :SYSLOG, so use REPORT to obtain information about it), is helpful to monitor the actions of users with the Superuser privilege. The advantage of Superuser logging is that you can get the information you need without producing a voluminous SYSLOG file.

CON0 logging is useful whether or not you have a hardcopy system console because you can search the file for pertinent text strings, easier than scanning pages of sometimes barely legible printout. Suggestion: turn CON0 logging off when you want to run a system utility on the system console (and turn it on again when you are done) to avoid logging meaningless I/O.

NOTE: To view the CON0 log file, stop logging in order to flush the system buffers. This will ensure that the log file is up to date. Then use the CLI TYPE command or the BROWSE or DISPLAY utility to view it. But do not view the log at the system console while CON0 logging is enabled or you will get into an infinite loop (since you will never get to the end of the file). If that happens, interrupt TYPE or BROWSE by typing CTRL–C, CTRL–A; interrupt DISPLAY by typing CTRL–C, CTRL–B.

If you are the system operator (PID 2) or have the System Manager privilege turned on, you can use this command to control system logging. You can check logging status, or start or stop system, Superuser, or CON0 logging, and move logged information from the system log file, from the error log file, or from the CON0 log file to other files.

If you will normally use system logging, you should include this command in your system's UP.CLI macro. For more details on logging, see *Managing AOS/VS and AOS/VS II.*

Command Switches

The section "Universal CLI Switches," earlier in this chapter, describes switches /1, /2, /L, /L=pathname, and /Q, which you can use with all commands. That section also explains the CLI32 switches /STR= and /ESTR=.

/CON0/START *[filename]*	CLI32 only. If you include a filename argument, this switch renames the current :CON0_LOG to that filename, then starts logging in a new :CON0_LOG file. If you do not include a filename, the CLI starts logging in the current :CON0_LOG file. You cannot use other switches with these switches.
/CON0/STOP	CLI32 only. Stops CON0 logging. You cannot use other switches with these switches.
/DETAIL=	Includes MINIMAL or FULL user account information in the system log file. The default is /DETAIL=MINIMAL. /DETAIL=FULL adds user access and security related data to the log.

SYSLOG (continued)

filename	If you specify a filename without switches, renames :SYSLOG to this name. If system logging was on, restarts :SYSLOG with the level of detail that was previously in effect.
/NOSOFTTAPEERRORS	Does not record soft tape errors in the :ERROR_LOG. This is the default action.
/RENAMEERROR	Renames :ERROR_LOG to the specified filename, and then continues error logging in a new :ERROR_LOG file.
/SOFTTAPEERRORS	Records soft tape errors in :ERROR_LOG.
/START	If you include a filename argument, this switch renames the current :SYSLOG to that filename, then starts logging in a new :SYSLOG file. If you did not include a filename, the CLI starts logging in the current :SYSLOG file.
/STOP	Stops system and Superuser logging.
/SUPERUSER/START	CLI32 only. Starts Superuser logging in the current :SYSLOG file. You must start system logging before starting Superuser logging. Therefore, you cannot use other switches with these switches.
/SUPERUSER/STOP	CLI32 only. Stops Superuser logging. You cannot use other switches with these switches.
/VERBOSE	CLI32 only. Displays status of system logging, Superuser logging, and CON0 logging.

SYSLOG Example 1

```
) SYSLOG ↵
OFF

) SYSLOG/START ↵
```

The first command displays the current system log setting. The second command starts logging in the current :SYSLOG file.

SYSLOG Example 2

```
) SYSLOG/START  9_DEC_91.LOG ↵

) SUPERUSER ON ↵
Su) LOGEVENT New syslog started 9–DEC–91 ↵
```

The first command starts system logging in a new :SYSLOG file after renaming the old log file to 9_DEC_91. The LOGEVENT command writes an entry into the new system log.

SYSLOG Example 3

```
Su) SYSLOG ↵                        (Displays logging status.)
ON                                  (Logging is on.)

Su) XEQ  REPORT  :SYSLOG ↵          (Displays report from the current log
                                    file on the console.)

Su) SYSLOG/START DEC.11.LOG ↵       (Renames :SYSLOG to DEC.11.LOG and
                                    starts logging to a new :SYSLOG.)
```

SYSLOG Example 4

```
Su) SYSLOG/VERBOSE ↵                (Displays complete logging status.)
SYSLOG off                          (System logging, Superuser logging, and
SYSLOG Superuser logging off        CONO logging are off.)
CON0 logging off

Su) SYSLOG/CON0/START ↵             (Starts logging CON0 I/O to the CON0 log file,
                                    :CON0_LOG.)
```

SYSLOG Example 5

The following example shows the status displayed with all options chosen.

```
Su) SYSLOG/V ↵
SYSLOG on, Full detail, Exclusion bit map set
SYSLOG Superuser logging on, Exclusion bit map set
CON0 logging on
```

SYSLOG Example 6

The following example shows that you must start system logging before you can start Superuser logging.

```
Su) SYSLOG/V ↵
SYSLOG off
SYSLOG Superuser logging off
CON0 logging off
Su) SYSLOG/START ↵
Su) SYSLOG/SUPERUSER/START ↵
```

!SYSTEM *Pseudomacro*

Expands to the operating system type: AOS, AOS/VS, or AOS/VS II.

Format

[!SYSTEM]

Without the /SPECIFIC switch, this pseudomacro returns the type of operating system: AOS/VS (which means AOS/VS or AOS/VS II) or AOS. (AOS is the Data General advanced operating system for ECLIPSE 16–bit computers.) The /SPECIFIC switch differentiates between AOS/VS and AOS/VS II.

- No arguments.
- No macro name switches.
- Requirement: *Standard*.

Why Use It?

Use the !SYSTEM pseudomacro to distinguish AOS/VS from AOS.
Use [!SYSTEM/SPECIFIC] to distinguish AOS/VS from AOS/VS II.

Switches

/SPECIFIC	CLI32 only. Expands to the text "AOS/VS" if the operating system revision level is less than 2100, or "AOS/VSII" if the level is greater.

!SYSTEM Example 1

(within a macro)

```
write Operating system is,,[!system]
```

This macro statement reports the current operating system (perhaps in a transcript file). The statement will look like this:

Operating system is AOS/VS

!SYSTEM Example 2

(within a macro)

```
write Operating system is,,[!system/specific]
```

This macro statement reports the current operating system (perhaps in a transcript file). The statement will look like this:

Operating system is AOS/VS or *Operating system is AOS/VSII*

!SYSTEM Example 3

(within a macro)

```
[!equal, [!system], AOS]
          xeq  myprog16
          .
[!end]
[!equal, [!system], AOS/VS]
          xeq  myprog32
          ...
[!end]
```

This macro uses the !SYSTEM pseudomacro to ascertain the current operating system, and then it executes either a 16-bit or 32-bit version of a program.

TAR_VS

Utility

Dumps files from the working directory in UNIX tar format or loads from a tar–format dump file.

Formats

To dump files:

```
TAR_VS/CREATE/FILE=dumpfilename   [ [!FILENAMES template] ... ]
                                  [ filename [...]            ]
```

To load files:

```
TAR_VS/XTRACT/FILE=dumpfilename   [ [!FILENAMES template] ... ]
                                  [ filename [...]            ]
```

To list without loading:

```
TAR_VS/TELL/VERBOSE/FILE=dumpfilename
```

- *To create dump (archive) files,* use the first format. The TAR_VS utility produces and reads dump files in the tar Archive/Interchange File Format specified in IEEE Std. 1003.1–1988. Primarily, TAR_VS is intended to let you transfer files between an AOS/VS or AOS/VS II system and a UNIX system — perhaps a DG/UX system running on an AViiON workstation.

  As arguments, you must use the !FILENAMES pseudomacro or the literal filename (be sure to use uppercase); the utility does not accept pathname arguments. You can use !FILENAMES with a template: for example,

  ```
  TAR_VS/CREATE/V/FILE=@MTJ0:0 [!FILENAMES MYPROG.+] )
  ```

  Unlike UNIX tar, TAR_VS does not interpret a directory name to refer to the files it contains and, recursively, those in its subdirectories. To dump the contents of a directory and its subdirectories, use a !FILENAMES template similar to the following:

  ```
  TAR_VS/CREATE/V/FILE=@MTJ0:0 [!FILENAMES MYDIR:#] )
  ```

  For dumping or loading, the dump filename can be a tape unit devicename (for example, @MTB0:0) or if you want to dump to disk, a disk filename.

- *To load a dump (archive) file,* use the second format. TAR_VS can read a dump file created by the tar utility on a UNIX system or by TAR_VS. It maintains the directory structure, if any, in the dump file. Unless you specify arguments (using !FILENAMES or literally), it loads all files in the dump file. If the dump file contains any lowercase filenames and you specify arguments, the files with lowercase names will not be loaded. To load files with lowercase filenames, omit filename arguments.

- *To list a dump (archive) file,* use the third format. TAR_VS displays file information without actually loading files.

NOTE: TAR_VS does not load conditionally based on date/time last modified; it has no analog to the LOAD_II /RECENT switch. If a file in the dump file has the same name as a file in the working directory, the utility deletes the file in the working directory and loads the file from the dump file. If you are unsure about whether the dump file has files that may replace files you want, use the third form of the command, TAR_VS/TELL/VERBOSE to learn filenames, sizes, and date/time last modified without loading files. This information, compared to information on files in the working directory, may help you decide which files to specify.

To confirm the dumping or loading of each file, include the /WAIT switch. Depending on the operation, the utility will prompt *add xxx* or *extract xxx*; respond Y to have it dump/load the file or anything else to have it skip the file. On a dump, to signify the end of the file list, press CTRL–D.

The TAR_VS utility converts characters and access permissions as detailed later. It reports disk blocks in 512–byte quantities (as usual for AOS/VS and AOS/VS II); pathnames are limited to 256 characters.

If you try to load from a dump file that is not in tar format, the utility will display the message *This doesn't look like a tar archive* and prompt for the next tape file.

You can abbreviate a switch name to the smallest number of characters needed to identify it. In most cases this is one character. Characters in switches are not case sensitive.

- Accepts templates in !FILENAMES pseudomacro.
- Accepts utility switches (described later).
- Requirement: *Standard.* You need Execute access to the program file TAR_VS.PR in :UTIL. To dump files, you need Execute access to any directory from which you want to dump and Read access to any file you want to dump; to load or copy into a directory, you need Write and Execute (WE) access to the directory.
- See also: CPIO_VS.

Why Use It?

Use TAR_VS to create tar–format dump files to be read by the tar utility on a UNIX system, or use it to load files dumped by tar on a UNIX system. If you are familiar with UNIX, you may elect to use the TAR utility instead of TAR_VS. (If you use TAR, you can use familiar UNIX syntax but must use AOS/VS–AOS/VS II device names; also, be aware that TAR provides only those features that TAR_VS does.)

Generally, use TAR_VS to dump or load more than one file (although it does work for individual files). For individual files, use the CPIO_VS utility.

Generally, to create dump files to be read on an AOS, AOS/VS, or AOS/VS II system, use the DUMP_II utility, not TAR_VS or CPIO_VS. For an RDOS system, use the RDOS utility.

Upper- and Lowercase Pathnames

In UNIX, filenames are case sensitive; for example, the names myfile and MYFILE indicate different files. Lowercase filenames are customary on UNIX systems. When you use TAR_VS to dump files, it dumps the names in precisely the case you specify. If you do not specify directly but use the !FILENAMES pseudomacro, the filenames will be dumped in uppercase (as returned by the CLI from !FILENAMES). For example, the command TAR_VS/CREATE...myfile dumps MYFILE with its name in lowercase; the command TAR_VS/CREATE...[!FILENAMES MYF+] dumps MYFILE and any other matching files with their names in uppercase. To display the case of the names under which files are dumped, use the /VERBOSE switch.

When loading, if you specify a template or pathname, the program will look for names with precisely the case you specify. It will convert filenames to uppercase as it loads them. The switches /VERBOSE/TELL show the case of filenames in the dump file without loading the files.

Conversion During Dumping or Loading

While dumping, TAR_VS converts certain characters and identifiers so that the dump file will load correctly on a UNIX system. While loading (extracting), the program converts other characters and identifiers so that the dump file will load correctly on an AOS/VS or AOS/VS II system.

Pathname Conversion

When TAR_VS creates a dump file, it converts AOS/VS and AOS/VS II pathname characters to UNIX pathname characters as follows.

AOS/VS II Character	UNIX Character	Example
: (directory)	/ (directory)	MYDIR:MYFILE becomes MYDIR/MYFILE
$ (dollars)	_ (underscore)	MY$FILE becomes MY_FILE
? (question mark)	_ (underscore)	MY?FILE becomes MY_FILE

All pathname characters are dumped in uppercase.

When TAR_VS reads a dump file (extracts), it converts UNIX pathname characters to AOS/VS and AOS/VS II pathname characters as follows.

UNIX Character	AOS/VS II Character	Example
/ (directory)	: (directory)	MYDIR/MYFILE becomes MYDIR:MYFILE
, (comma)	? (question mark)	MYFILE,01 becomes MYFILE?01
– (hyphen)	? (question mark)	MY–FILE becomes MY?FILE

The operating system converts lowercase characters in pathnames to uppercase; for example, myfile becomes MYFILE.

Group ID (GID) and User ID (UID)

While dumping, TAR_VS provides a group identification number (GID) and a user identification number (UID) for each file. TAR_VS sets the GID of each file to 0, and attempts to look up the UID of the file owner.

UNIX systems, which keep track of users through UID numbers, store UIDs in a file whose path is normally /etc/passwd. The AOS/VS equivalent of this path is :etc:passwd, so TAR_VS searches this path.

AOS/VS and AOS/VS II systems identify users by usernames, not numbers. Entries in the :etc:passwd file (if it exists) correlate usernames with UID numbers. When you dump a file, if the owner's username exists in file :etc:passwd, then TAR_VS writes the corresponding UID to the dump file along with the file. If :etc:passwd has no entry for the file owner, then TAR_VS writes a −1 to the archive as the UID.

Generally, when you dump from an AOS/VS or AOS/VS II system, there is no password file in :etc:passwd, so the User IDs on files in the dump file are −1.

Access Control List (UNIX Permissions)

In a dump, TAR_VS sets the dump file access control list (ACL) to OWR for the owner and R for other users. When TAR_VS dumps each file, it converts the ACL to UNIX permissions, so far as possible (it can convert the Owner and Other fields only). For example, if the ACL of a nondirectory file is username,OWARE +,E, TAR_VS will convert the ACL so that after loading on a UNIX system, its permissions will be –rwx– – – – – –x.

Link Files

When TAR_VS creates a dump file, it resolves links and copies the actual resolution file to the dump file.

Time Last Modified and Time Last Accessed

When TAR_VS creates a dump file, it dumps the existing time last modified and time last accessed along with each file. When the program loads from a dump file, it changes the time last modified and time last accessed to the time of the load. (DUMP/DUMP_II and LOAD/LOAD_II behave the same way.)

TAR_VS does not support the UNIX tar −m option.

TAR_VS Utility Switches

/BLOCK=n	Tells the program to write data at n disk blocks per record. This is analogous to the /BUFFERSIZE= switch. The default and maximum number is 32 blocks (16 Kbytes) per record. On a load (extract), the program determines the block size automatically.
/FILENAME=pathname	Tells the utility to use pathname as the dump file (for example, @MTB0:0 or MY_DUMPFILE).
/LINK	On loads (extracts) only, tells the program to report if it cannot resolve links to the file being loaded. If you omit this switch, the utility does not report unresolved links.
/OWNER	On loads (extracts) only, tells the program to update the ownership field to your username instead of the owner included in the dump file.
/R	Tells the program to write files to the end of the dump file. Use this instead of TAR_VS/CREATE if you decide to add filenames to the end of the dump file. (The /CREATE switch tells the program to start at the beginning of the dump file; for tape, this overwrites all material on that file.) Some cartridge tape units, such as 139–Mbyte, 1/4–inch tape units, cannot support the /R switch.
/VERBOSE	Tells the program to display the names of files dumped (preceded by a –, for append) or loaded (preceded by x –, for extract). You can use this with any form of the utility command line.
/WAIT	Tells the program to display filenames but not load them. Use this for dumps loads for interactive operation. It is analogous to the LOAD/LOAD_II switch /CONFIRM.

TAR_VS Example 1

```
) TAR_VS/CREATE/FILENAME=@MTJ0:0/VERBOSE myprog.c⮐
a– myprog.c 21 blocks
)
```

This example shows TAR_VS dumping file MYPROG.C. The filename for the dump file is the first file of the tape on unit MTJ0. The program dumps the file and verifies this with *–a* (for append), the filename, and the file size (21 disk blocks). The filename under which MYPROG.C is dumped is lowercase: myprog.c.

The tape can be taken to a UNIX system for loading using tar –xtract. The output file (/FILENAME=) can be a disk file, which can be copied over a network to a UNIX system; then it, too, can be loaded with a tar –xtract command.

!UDIVIDE

Pseudomacro

Expands to the quotient of an argument divided by another.

Format

```
[!UDIVIDE  [ integer1 integer2  ]  ]
           [ integer [ ... ]¹    ]
```

¹ CLI32 only.

This pseudomacro divides one argument by another argument and returns the integer quotient. If you provide only one argument, that argument is returned as the answer. If you specify only a space as the single argument, 0 is returned as the answer. If you specify no argument and no space, 1 is returned as the answer. If you specify 0 as the divisor, the CLI returns the message, "Error: Zero divisor."

With CLI16, you can use two unsigned integer arguments; the first argument may be double precision (in the range 0 through 4,294,967,295), but the second must be single precision (in the range 1 through 65,535).

With CLI32, you can use as many arguments as you want (up to the limit on line length). Each argument must be an unsigned decimal integer value in the range 0 through 4,294,967,295. The CLI will process arguments sequentially and pass the quotient to the next integer for division.

- No templates.
- No argument switches.
- No macro name switches.
- Requirement: *Standard.*
- See also: !UADD, !UMODULO, !UMULTIPLY, !USUBTRACT, VAR and !VAR (CLI32 only), VARn and !VARn.

Why Use It?

Use the !UDIVIDE pseudomacro to divide integer values and either display the resulting quotient or use it in a test or calculation. To display the remainder, use the !UMODULO pseudomacro.

!UDIVIDE Example 1

```
) WRITE [!UDIVIDE 10 3] )
3
```

This command displays the result of dividing the value 10 by the value 3.

!UDIVIDE Example 2

```
) WRITE [!UDIVIDE 100  5  10] ⤶
2
```

With CLI32, this command displays the result of dividing the value 100 by 5 and dividing that quotient by 10.

!UDIVIDE Example 3

The macro QUOTIENT.CLI contains the following commands:

```
var0 [!read Enter dividend: ,,,]
var1 [!read Divide it by: ,,,]
write The quotient is [!udivide,[!var0],[!var1]].
```

The first command prompts for the dividend value, assigning it to the CLI variable VAR0. The second statement requests the divisor value, assigning it to VAR1. The third statement uses !UDIVIDE to perform the division, and then displays a message with the resulting quotient. Sample dialog is

```
) QUOTIENT⤶
Enter dividend:  36 ⤶
Divide it by:  7⤶
The quotient is 5.
```

As another example, see the CALCULATOR macro in Chapter 4.

```
) WRITE [!UMINIMUM [!LENGTH of a moderate string], 5 1]⤶
8
```

In this last example, the "!LENGTH of a moderate string" expands to the lengths of all arguments: 2 (for "of"), 1 (for "a"), 8 (for "moderate"), and 6 (for "string"). Of these arguments, and the 5 and 1, the minimum is 1, which is what the CLI displays.

```
comment Macro COMPARE.CLI — compares two files and displays
comment length of the shorter.
var0 [!size %1%]
var1 [!size %2%]
write The shorter file holds [!uminimum [!var0],[!var1]] bytes.
```

This macro compares the sizes of two files.

!UMODULO *Pseudomacro*

Expands to the value of the first argument given a modulus specified by the second argument.

Format

```
[!UMODULO  ⎡ integer1 integer2 ⎤ ]
           ⎣ integer [ ... ]¹  ⎦
```

¹ CLI32 only.

This pseudomacro divides one unsigned decimal integer argument by another and returns the remainder. If you provide only one argument, that argument is returned as the answer. If you specify only a space as the single argument, 0 is returned as the answer. If you specify no argument and no space, 4294967295 is returned as the answer. If you specify 0 as the argument, 0 is returned as the answer.

With CLI16, the first integer may be double precision (in the range 0 through 4,294,967,295), but the second must be single precision (in the range 1 through 65,535). With CLI32, you can use more than two arguments; all integers may be double precision.

The pseudomacro evaluates the first argument according to the modulus specified by the next, and then returns the integer result. This value is equivalent to the remainder produced by dividing the first argument by the second. CLI32 evaluates a string of integers left to right (as, for example, [!UMODULO 100 21 6], which returns 4).

- No templates.
- No argument switches.
- No macro name switches.
- Requirement: *Standard.*
- See also: !UDIVIDE.

Why Use It?

Use the !UMODULO pseudomacro to perform modular arithmetic or obtain the remainder for a division.

!UMODULO Example 1

```
) WRITE [!UMODULO 10 3] ↵
1
```

This command displays on the screen the modular arithmetic result of the value 10 modulo 3.

!UMODULO Example 2

The macro DIVIDE.CLI contains the following commands:

```
write The quotient is [!udivide, %1%, %2%]
write The remainder is [!umodulo, %1%, %2%]
```

The first statement divides the first macro argument by the second, and then displays the integer quotient result. The second statement uses modular arithmetic to obtain and display the remainder for the division of the two arguments. Sample dialog is

```
) DIVIDE 37 5 ↵
The quotient is 7
The remainder is 2
```

As another example, see the CALCULATOR macro in Chapter 4.

!UMULTIPLY

Pseudomacro

Expands to the product of integers.

Format

```
[!UMULTIPLY  [ integer1 integer2    ]  ]
             [ integer [ ... ]1     ]
```

1 CLI32 only.

This pseudomacro multiplies one integer argument by another and returns the product. If you provide only one argument, that argument is returned as the answer. If you specify only a space as the single argument, 0 is returned as the answer. If you specify no argument and no space, 1 is returned as the answer. If you specify 0 as the argument, 0 is returned as the answer.

If the product is greater than 4,294,967,295, the returned value is the product modulo 4,294,967,296; that is, the remainder produced by dividing the product by 4,294,967,296.

With CLI16, you can use two unsigned integer arguments; the first argument may be double precision (in the range 0 through 4,294,967,295), but the second must be single precision (in the range 1 through 65,535).

With CLI32, you can use as many arguments as you want (up to the limit on line length). Each argument must be an unsigned decimal integer value in the range 0 through 4,294,967,295. The CLI will process arguments sequentially and pass the product to the next integer for multiplication.

- No templates.
- No argument switches.
- No macro name switches.
- Requirement: *Standard.*
- See also: !UADD, !UDIVIDE, !UMULTIPLY, !USUBTRACT, VAR and !VAR (CLI32 only), VARn and !VARn.

Why Use It?

Use the !UMULTIPLY pseudomacro to multiply integer values and either display the resulting product or use it in a test or calculation.

!UMULTIPLY Example 1

```
) WRITE [!UMULTIPLY 256 12] ↵
3072
```

This command displays the result of multiplying the value 256 by the value 12.

UNLOCK *Command*

Frees a locked CLI.

Format

$$\text{UNLOCK} \begin{bmatrix} \textit{CLI-command-to-enable}^{1} \\ \text{/CX} \textit{[EXEC-command-to-enable]}^{1} \end{bmatrix}$$ 1 CLI32 only.

This command frees a CLI process that was locked with the LOCK command. The LOCK command works only in CLI32 or LOCK_CLI.PR.

In CLI32, you can enter specific commands to unlock. If you omit arguments, the command applies to those commands that relate to security, including the following:

BLOCK	DEBUG	JPRELEASE	QFTA	SUPERUSER
BYE	DELETE	MOVE	QSUBMIT	TERMINATE
CHAIN	EXECUTE	PRIVILEGE	RELEASE	XEQ
CONNECT	INITIALIZE	PROCESS	RENAME	
COPY	JPINITIALIZE	QBATCH	SUPERPROCESS	

NOTE: The DUMP and LOAD commands do not work with a locked CLI since they invoke the PROCESS command, which is locked.

In CLI32, if you include the /CX switch without arguments, the CLI enables all EXEC commands (explained in the EXEC chapter, *Managing AOS/VS and AOS/VS II*). If you include arguments, the CLI enables those EXEC commands only; for example, UNLOCK/CX SPOOLSTATUS unlocks the EXEC SPOOLSTATUS command. Unlocking EXEC commands is most useful for the master CLI that runs on the system console.

After you enter the UNLOCK command, the CLI requires a password (unless you used the /STATUS switch). CLI32 will prompt for the current password; LOCK_CLI will not ask anything. You must enter the correct password exactly, without using any edit keys or Screenedit commands, which are interpreted literally as part of your response. The system does not echo your entry.

If you enter the wrong password, CLI32 will display an error message; LOCK_CLI will display no response but returns the prompt,). If you entered the correct password, the CLI becomes unlocked; otherwise it remains locked.

Setting the password of CLI32 is explained under the PASSWORD command; setting the password of LOCK_CLI is explained in *Managing AOS/VS and AOS/VS II.*

- No templates.
- No argument switches.
- Requirement: *Standard.*
- See also: LOCK, PASSWORD (CLI32 only).

UNLOCK (continued)

Why Use It?

Use the UNLOCK command to enable the commands that were disabled by a previous LOCK command. You must do this to perform system operator functions or any other disabled operation. If your system UP.CLI macro includes commands that lock the CLI that PID 2 is running, you must use the UNLOCK command before calling your system DOWN.CLI macro.

Command Switches

The section "Universal CLI Switches," earlier in this chapter, describes switches /1, /2, /L, /L=pathname, and /Q, which you can use with all commands. That section also explains the CLI32 switches /STR= and /ESTR=.

/ALL	(CLI32 only.) Enables (unlocks) all CLI commands. Use this switch when the CLI was locked with LOCK/ALL. Combine with /CX when the CLI was locked with LOCK/CX/ALL and you want to unlock all EXEC commands.
/CX	Enables (unlocks) commands to EXEC (CONTROL @EXEC commands). If you include one or more EXEC commands as arguments, the CLI unlocks those commands.
/STATUS	(CLI32 only.) Without an argument, displays the names of all unlocked non−EXEC commands. To determine which EXEC commands are unlocked, use this switch with /EXEC. When you supply an argument, displays the names of those specified command(s) that are unlocked. Does not require password.
/V	(CLI32 only.) Displays the names of the commands you are unlocking. Requires password. If the CLI is currently unlocked, displays the list of default UNLOCK commands (listed in the "Format" section of this command description).
/VERIFY	(CLI32 only.) Same as /V.

UNLOCK Example, LOCK_CLI

The following example assumes the password is XXX.

```
) XEQ LOCK_CLI ⤶              (Runs the LOCK_CLI program.)

) SUPERUSER ON ⤶              (Tests the Superuser command.)
)                             (LOCK_CLI ignores the command.)

) UNLOCK ⤶                    (Starts to unlock the CLI.)
XXX ⤶                         (Types password — it does not echo.)

) SUPERUSER ON ⤶              (Tries to turn Superuser on.)
Su)                           (CLI obeys the command.)
```

UNLOCK Example, CLI32

```
) LOCK↵
Password: ABC↵                 (Password does not echo.)
)
) XEQ  MYPROG↵                 (Tries to execute a program.)
Error: Command is locked, XEQ  (CLI displays error message.)

) UNLOCK/STATUS XEQ↵           (Check the command unlock status.)
)                              (No response indicates that XEQ is locked.)

) UNLOCK↵                      (Unlock all locked commands.)
Password: ABC↵                 (Password does not echo.)

) UNLOCK/STATUS XEQ↵           (Confirm that the XEQ command is unlocked.)
XEQ                            (Display confirms that it is unlocked.)

) XEQ  MYPROG↵                 (MYPROG executes.)
...
```

!USERNAME *Pseudomacro*

Expands to the username of the person running the CLI.

Format

[!USERNAME]

This pseudomacro represents the username of the CLI process.

- No arguments.
- No macro name switches.
- Requirement: *Standard*.
- See also: !PID, WHO.

Why Use It?

Use the !USERNAME pseudomacro to include the CLI username in a command argument. If you write macros for use by other users, you may need to include the log–on directory name in a macro. You can do this with :UDD:[!USERNAME].

!USERAME Example 1

```
) WRITE Call me [!USERNAME]. ↵
Call me ISHMAEL.
```

This command reports the username of the person running this CLI.

!USERNAME Example 2

(within a macro)

```
send %1% This message was sent by [!username].
```

This command sends a message that includes the sender's username. The message will look something like this:

From PID 68: This message was sent by ZONIS.

!USERNAME Example 3

(within a macro)

```
.
delete/2=ignore  :udd:[!username]:batch.–
qbatch/quoutput=:udd:[!username]:batch_output.out&
/qlist=:udd:[!username]:batch_output.list  %1–%
.
```

This macro runs a batch job and places the batch output and batch list files in the user's log–on directory. To determine the outcome of the job, the user can then type the output file instead going to the line printer (the default output file).

!USUBTRACT *Pseudomacro*

Expands to the difference between integer values.

Format

$$[\text{!USUBTRACT} \left\{ \begin{array}{l} \text{integer1 integer2} \\ \text{integer integer } [\ \dots\ ]^{1} \end{array} \right\}]$$

1 Multiple arguments in CLI32 only.

This pseudomacro subtracts the value of one argument from another and returns the result. Each argument must be an unsigned decimal integer value in the range 0 through 4,294,967,295.

With CLI16, you can use two arguments; with CLI32, you can use as many arguments as you want (up to the limit on line length). The CLI will pass the subtrahend of one operation to the next for subtraction.

If the difference between the two values is negative (the first argument is less than the second), the result is the absolute value of the difference modulo 4,294,967,296; that is, the remainder produced by subtracting the absolute value of the difference from 4,294,967,296.

- No templates.
- No argument switches.
- No macro name switches.
- Requirement: *Standard.*
- See also: !UADD, !UDIVIDE, !UMULTIPLY, VAR and !VAR (CLI32 only), VARn and !VARn.

Why Use It?

Use the !USUBTRACT pseudomacro to subtract one number from another and either display the result or use it in a test or calculation.

!USUBTRACT Example 1

```
) WRITE [!USUBTRACT 17 5]⤶
12
```

This command displays on the result of subtracting the value 5 from the value 17.

!USUBTRACT Example 2

```
) WRITE [!USUBTRACT 5000 199 25]⤶
4776
```

With CLI32, this command displays the result of subtracting from 5000 the values 199 and 25.

!USUBTRACT Example 3

```
) WRITE [!USUBTRACT 5 17]↵
4294967284
```

This command displays on the screen the result of subtracting the value 17 from the value 5. The displayed result shows the CLI's way of displaying a negative number. Also, the CLI would add (that is, !uadd) 4,294,967,284 to 12 to obtain a result of 0.

!USUBTRACT Example 4

The macro DIFF.CLI contains the following lines.

```
comment This is macro DIFF.CLI.
[!uge, %1%, %2%]
            write The difference is [!usubtract, %1%, %2%]
[!else]
            write The difference is -[!usubtract, %2%, %1%]
[!end]
```

This macro compares the two arguments given it. If the first argument is greater than or equal to the second, the macro subtracts the second value from the first and displays the first message. Otherwise, the macro subtracts the first argument from the second, and displays it as a negative value. Sample dialog is

```
) DIFF  25 4↵
The difference is 21

) DIFF  4 25↵
The difference is -21
```

As another example, see the CALCULATOR macro in Chapter 4.

VAR *Command*

Sets or displays an integer value (CLI32 only — VARn, for both CLIs, follows !VAR).

Format

VAR *[integer–or–expression–with pseudomacros]*

The CLI32 VAR command offers access to an unlimited number of variables. You can set and display variable values using VAR/NAME= commands; you can also display values using the !VAR/NAME= pseudomacro. For example,

```
) VAR/NAME=FIRST_MUSTANG 1965 ⤶
) VAR/NAME=FIRST_MUSTANG ⤶
1965
```

The VAR commands create a variable named FIRST_MUSTANG and store the integer 1965 in it.

The argument to the VAR command can be an integer or expression — including VAR pseudomacros — that evaluates to an integer expression. For example.

```
) VAR/NAME=ARG1 10 ⤶
) VAR/NAME=ARG2 2 ⤶
) VAR/NAME=QUOTIENT [!UDIVIDE [!VAR/NAME=ARG1 [VAR/NAME=ARG2]] ⤶
) VAR/NAME=QUOTIENT ⤶
5
```

The VAR command exists in addition to — not as a replacement of — VARn commands. The VAR/NAME= variables and the VARn variables simply differ. For example, the following VAR and VAR0 commands assign values to two different variables that happen to have the same name (VAR0) in different contexts.

```
) VAR/NAME=VAR0  83 ⤶
) VAR0  38 ⤶
```

After you assign a value to a variable, if you descend to another environment level, the variable retains the assigned value on the new level. If you have not assigned a value to a variable, the CLI assumes its value is 0.

- No templates.
- Requirement: *Standard*.
- See also: !VAR, VARn, STRING.

Why Use It?

In CLI32, use the VAR command with the /NAME switch to provide meaningful names for your variables. You are not limited to the 10 VARn names (VAR0, VAR1, ..., VAR9). With CLI16, use the VARn variables.

VAR Command Switches

The section "Universal CLI Switches," earlier in this chapter, describes switches /1, /2, /L, /L=pathname, /Q, /STR=, and /ESTR=.

/ALL

Displays the values of all the variables that you have specified at the current environment level. You can use this switch along with the /INFO, /KILL, /LEVEL, and /PREVIOUS switches.

/INFO

Displays the value of of all variables you have assigned. With no other switches, /INFO displays the values of all variables at the current environment level. For example,

```
) VAR/NAME=HOLDS_100 100 ↵
) VAR/NAME=HOLDS_200 200 ↵
) VAR/INFO ↵
HOLDS_100 = 100
HOLDS_200 = 200
```

Used together with /ALL, /INFO displays the values of all defined variables. The display shows the number of the environment level in parentheses. When more than one level uses the same variable, the repeated values do not appear. For example,

```
) VAR/NAME=A  10000 ↵
) PUSH ↵
) VAR/NAME=A  11111 ↵
) VAR/NAME=B  20000 ↵
) VAR/INFO/ALL ↵
A (0) 10000
A (1) 11111
B (1) 20000
) VAR/ALL/PREVIOUS ↵
) VAR/INFO/ALL ↵
A (0) 10000
```

For information on a specific level, use /INFO/LEVEL=.

For information on a specific string, use /INFO/NAME. The display includes the value of the specified variable at all environment levels — again, as with /ALL, with the level shown in parentheses.

/KILL

Clears (sets to null) a named variable. If you include /ALL and omit /NAME= (/KILL/ALL), the CLI clears all variables *on the current level only*. (This differs from the /INFO/ALL, which displays values on all levels.) If you really want to clear all variables, you can chain to another CLI (CHAIN :CLI).

WRITE Command Switches

The section "Universal CLI Switches," earlier in this chapter, describes switches /1, /2, /L, /L=pathname, and /Q, which you can use with all commands. That section also explains the CLI32 switches /STR= and /ESTR=.

/7BIT

(CLI32 only.) Tells the CLI to remove all parity bits on the output of the command. The CLI generates an error when it expands an !ASCII argument to a special character such as an angle or square bracket, or parenthesis. Add a high order (parity) bit to the !ASCII argument to prevent the CLI from interpreting it, and with this switch produce 7-bit output of the desired character. For example, the following command produces an error:

```
) WRITE/L=new.cli,WRITE,[!ASCII 133]!DATE[!ASCII 135] ↵
Error: Unmatched [ ( or <, expanding !ASCII 133.
```

Changing the ASCII codes to 333 and 335 and adding /7BIT to WRITE prevents the error so that a functional macro is created.

```
) WRITE/7BIT/L=new.cli,WRITE,[!ASCII 333]!DATE[!ASCII 335] ↵
)NEW ↵
26-Jun-92
```

FILEID=file–ID

(CLI32 only.) Tells the CLI to write the text of the argument(s) to the file identified by file–ID. You must include the file–ID in the WRITE command line, and the file must have been opened for writing (OPEN/WRITE pathname). If you omit this switch (and the /L switch), the CLI will display all arguments on the terminal. You can also use /FORCE to write the message to disk immediately.

The default file ID is the file's name (not pathname), without any trailing suffix. You can learn the file IDs of all open files by typing the OPEN command without an argument.

/FORCE

(CLI32 only.) With the /FILE switch, forces the system to place data in the file after every WRITE/FILE command. Otherwise, the system temporarily stores information from WRITE/FILE commands (in a buffer) and periodically writes the accumulated information into filename. /FORCE is valid only with the /FILEID switch.

/NONEWLINE

(CLI32 only.) Tells the CLI to omit the NEW LINE terminator that it normally writes at the end of each line. This switch lets you use different WRITE commands to write text to the same line in a file, as in the following example:

```
) WRITE/L=MYFILE/NONEWLINE/Hello t ↵
) WRITE/L=MYFILE here ↵
Hello there
```

The /NONEWLINE also lets you create a file with control or other special characters (for example, as shown in the QPRINT description, you can create a file of one byte containing a 377 using a command of the form WRITE/L=filename/NONEWLINE [!ASCII 377]. You can use any ASCII value instead of 377.

Using WRITE with Special Characters

The following list of octal values, when used with the [!ASCII] pseudomacro, results in the display of the corresponding characters and special effects.

This technique works by adding 200 to the ASCII value of characters, thus masking the real characters. It will produce the following results only with terminals using seven-bit character sets and U.S. standard keyboards.

203	Enable blink	230	Cursor right	257	/
204	Disable blink	231	Cursor left	260−271	0−9
207	Bell	232	Cursor down	272	:
210	Home	234	Start dim	273	;
211	Tab	235	Stop dim	274	<
212	New line	240	Space	275	=
213	Erase EOL	241	!	276	>
214	Erase page	242	"	277	?
215	Carriage return	243	#	300	@
216	Start blink	244	$	301−332	A−Z
217	Stop blink	245	%	333	[
220,c,r	Set cursor position:[1]	246	&	334	\
	c−200 sets column;	247	’ (right apostrophe)	335	]
	r−200 sets row	250	(	336	^
222	Enable roll	251	)	337	_ (underscore)
223	Disable roll	252	*	340	‘ (left apostrophe)
224	Start underline	253	+	341−372	a−z
225	Stop underline	254	, (comma)	373	{
226	Start reverse video[2]	255	− (dash)	374	|
227	Cursor up	256	. (period)	375	}
				376	~

[1] The screen layout is a rectangular grid that begins in the upper left corner. The first of the numbers c,r specifies a column and the second specifies a row. Each number is offset from 200. Consequently, 220,200,200 specifies 0,0 true decimal, the upper left corner; 220,250,200 specifies 40,0 true decimal, the middle of the top row; 220,200,224 specifies 0,20 true decimal, near the bottom of the first column; 220,250,214 specifies 40,12 true decimal,near the center of the screen.

[2] Use [!ASCII 214] to stop reverse video.

For example, the following command clears the screen, moves the cursor near the middle left of the fresh screen, and displays *Hello there, friend!* with the first two words blinking.

```
) WRITE [!ASCII 214 220 200 214 216]&
Hello[!ASCII 240]there[!ASCII 217 254 240]friend!
```

See Example 6, which shows how the WRITE command with the [!ASCII] pseudomacro forms the skeleton of a macro that displays a menu of choices and validates the response.

Index

Document Set

For Users

AOS / VS and AOS / VS II Glossary (069–000231)

For all users, this manual defines important terms used in AOS/VS and AOS/VS II manuals, both regular and preinstalled.

Learning to Use Your AOS / VS System (069–000031)

A primer for all users, this manual introduces AOS/VS (but the material applies to AOS/VS II) through interactive sessions with the CLI, the SED and SPEED text editors, programming languages, Assembler, and the Sort/Merge utility. *Using the CLI (AOS and AOS / VS II)* is a good follow–up.

SED Text Editor User's Manual (AOS and AOS / VS) (093–000249)

For all users, this manual explains how to use SED, an easy–to–use screen–oriented text editor that lets you program function keys to make repetitive tasks easier. The *SED Text Editor* template (093–000361) accompanies this manual.

Using the AOS / VS System Management Interface (SMI) (069–000203)
Using the AOS / VS II System Management Interface (SMI) (069–000311)

For those working with preinstalled systems and those on regular systems who want an alternative to the CLI, the SMI is an easy–to–use, menu–driven program that helps with some file maintenance tasks.

Using the CLI (AOS / VS and AOS / VS II) (093–000646)

For all users, this manual explains the AOS/VS and AOS/VS II file and directory structure and how to use the CLI, a command line interpreter, as the interface to the operating system. This manual explains how to use the CLI macro facility, and includes a dictionary of CLI commands and pseudomacros.

For System Managers and Operators

AOS / VS and AOS / VS II Error and Status Messages (093–000540)

For all users, but especially for system managers and operators of regular systems, this manual lists error and status messages, their source and meaning, and appropriate responses. This manual complements *Installing, Starting, and Stopping AOS / VS*; *Installing, Starting, and Stopping AOS / VS II*; and *Managing AOS / VS and AOS / VS II.*

AOS / VS and AOS / VS II Menu-Based Utilities (093–000650)

A keyboard template to identify function keys. A number of system management programs—such as Disk Jockey, VSGEN, and the SMI—and the BROWSE utility use the function keys identified on this template.

Information Update: Starting Your ECLIPSE MV / 1000 DC (014–001728)

Updates *Starting and Updating Preinstalled AOS / VS* and *Starting and Updating Preinstalled AOS / VS II.*

Installing, Starting, and Stopping AOS / VS (093–000675)
Installing, Starting, and Stopping AOS / VS II (093–000539)

For system managers and operators of regular (as opposed to preinstalled) systems, these manuals explain the steps necessary to format disks, install a tailored operating system, create the multiuser environment, update the system or microcode, and routinely start up and shut down the system. *AOS / VS and AOS / VS II Error and Status Messages* and *Managing AOS / VS and AOS / VS II* are companions to these manuals.

Managing AOS / VS and AOS / VS II (093–000541)

For system managers and operators, this manual explains managing an AOS/VS or AOS/VS II system. Managing tasks include such topics as editing user profiles, managing the multiuser environment with the EXEC program, backing up and restoring files, using runtime tools, and so forth. This manual complements the "Installing" manuals, whether for regular or preinstalled systems.

Starting and Updating Preinstalled AOS / VS (069–000293)
Starting and Updating Preinstalled AOS / VS II (069–000294)

For those working with preinstalled (as opposed to regular) operating systems on all computers except ECLIPSE® MV/3000 DC and ECLIPSE MV/5000™ DC series systems, these manuals explain how to start, update, and change certain system parameters. The manuals also help you interpret error messages and codes. Companion manuals are *Using the AOS / VS System Management Interface* and *Using the AOS / VS II System Management Interface.*

Starting and Updating Preinstalled AOS / VS on ECLIPSE® MV / 3000 DC and ECLIPSE MV / 5000™ DC Series Systems (069–000481)
Starting and Updating Preinstalled AOS / VS II on ECLIPSE® MV / 3000 DC and ECLIPSE MV / 5000™ DC Series Systems (069–000480)

For those working with preinstalled (as opposed to regular) operating systems on ECLIPSE® MV/3000 DC and ECLIPSE MV/5000™ DC series computers, these manuals explain how to start, update, and change certain system parameters. The manuals also help you interpret error messages and codes. Companion manuals are *Using the AOS / VS System Management Interface* and *Using the AOS / VS II System Management Interface.*

If you have one of these computer systems, use the pertinent manual above; discard any other *Starting and Updating Preinstalled* manuals you receive.

Using the AOS / VS System Management Interface (SMI) (069–000203)
Using the AOS / VS II System Management Interface (SMI) (069–000311)

For those working with preinstalled systems and those on regular systems who want an alternative to the CLI, the SMI is an easy–to–use, menu–driven program that helps with system management functions and some file maintenance tasks.

For Programmers

AOS / VS, AOS / VS II, and AOS / RT32 System Call Dictionary, ?A through ?Q (093–000542)
AOS / VS, AOS / VS II, and AOS / RT32 System Call Dictionary, ?R through ?Z (093–000543)

For system programmers and application programmers who use system calls, this two–volume manual provides detailed information about system calls, including their use, syntax, accumulator input and output values, parameter packets, and error codes. *AOS / VS System Concepts* is a companion manual.

AOS / VS Debugger and File Editor User's Manual (093–000246)

For assembly language programmers, this manual describes using the AOS/VS and AOS/VS II debugger for examining program files, and the file editor FED for examining and modifying locations in any kind of disk file, including program and text files. The *AOS / VS Debug / FED* template (093–000396) accompanies this manual.

AOS / VS Link and Library File Editor (LFE) User's Manual (093–000245)

For AOS/VS and AOS/VS II programmers, this manual describes the Link utility, which builds executable program files from object modules and library files, and which can also be used to create programs to run under the AOS, MP/AOS, RDOS, RTOS, or DG/UX™ operating systems. This manual also describes the Library File Editor utility, LFE, for creating, editing, and analyzing library files; and the utilities CONVERT and MKABS, for manipulating RDOS and RTOS files.

AOS / VS Macroassembler (MASM) Reference Manual (093–000242)

For assembly language programmers, this reference manual describes the use and operation of the MASM utility, which works under AOS/VS and AOS/VS II.

AOS / VS System Concepts (093–000335)

For system programmers and application programmers who write assembly–language subroutines, this manual explains basic AOS/VS system concepts, most of which apply to AOS/VS II as well. This manual complements both volumes of the *AOS / VS, AOS / VS II, and AOS / RT32 System Call Dictionary.*

SPEED Text Editor (AOS and AOS / VS) User's Manual (093–000197)

For programmers, this manual explains how to use SPEED, a powerful (but unforgiving) character–oriented text editor.

Other Related Documents

AOS / VS and AOS / VS II Performance Package User's Manual (093–000364)

For system managers, this manual explains how to use the AOS/VS and AOS/VS II Performance Package (Model 30718), a separate product that is useful for analyzing and perhaps improving the performance of AOS/VS and AOS/VS II systems.

Backing Up and Restoring Files With DUMP_3 / LOAD_3 (093–000561)

For system managers, operators, and experienced users, this manual explains the DUMP_3/LOAD_3 product, separately available, which provides backup and enhanced restoration functions, including precise indexing of files on a backup tape set.

Configuring and Managing the High-Availability Disk-Array / MV (H.A.D.A. / MV) Subsystem (014–002160)

For system managers of the H.A.D.A./MV subsystem (a separate product), this manual explains how to configure, operate, and replace subsystem controllers, disk modules, and tape modules. This manual also explains how to replace fans, power supplies, and other subsystem hardware.

Configuring Your Network with XTS (093–00689)

For network administrators, managers, or operators responsible for designing, configuring, or maintaining a network management system, this manual describes how to manage and operate Data General's XODIAC™ Transport Service (XTS and XTS II) under AOS/VS and AOS/VS II.

Installing and Administering DG TCP / IP (093–701027)

For network managers and operators, this manual explains how to install and manage a TCP/IP network under AOS/VS.

Managing AOS / VS II TCP / IP (093–000704)

For network managers and operators, this manual explains how to install and manage a TCP/IP network under AOS/VS II.

Managing AOS / VS II ONC™ / NFS® Services (093–000667)

For network managers and operators, this manual explains how to install and manage an ONC Network File System server software under AOS/VS II.

Managing and Operating the XODIAC™ Network Management System (093–000260)

For network managers and operators, this manual describes how to install and manage the Data General proprietary network software.

Using CLASP (Class Assignment and Scheduling Package) (093–000422)

For system managers, this manual explains how to use the AOS/VS and AOS/VS II Class Assignment and Scheduling Package (Model 31134), a separate product that is useful for tailoring process scheduling to the needs of a specific site.

Using the Dump Tool (093–000519)

For experienced system programmers and operating system experts, this manual explains how to use the Dump Tool to find and display the values of locations in memory dump and break files.

Using the MV Data Center Manager (093–000769)

For system managers, this manual explains how to use the MV Data Center Manager software, a separate product that manages multiple ECLIPSE MV/Family computers from an AViiON workstation.