DataGeneral
Data General Corporation, Westboro, Massachusetts 01580

Customer Documentation

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

093-000646-01

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.

Printed in the United States of America
Rev. 01, December 1991
Licensed Material – Property of Data General Corporation
Ordering No. 093-000646

Notice

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

Revision History:

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

Effective with:

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.

Instructions for Inserting Dividers

We included tabbed dividers with your manual to make the information easier to access. Please insert the dividers as follows:

Divider Name	Insert Before Page
CLI Environment	3–1
Macros	4–1
Command Dictionary	5–1
A–D Commands and Utilities	5–21
E–L Commands and Utilities	5–155
M–R Commands and Utilities	5–271
S–Z Commands and Utilities	5–401
Magnetic Media	6–1

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 1 – Introducing the CLI

Chapter 2 – Using the File System

Chapter 3 – CLI Environment Settings

Chapter 4 – CLI Macros

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

Chapter 6 – Using Magnetic Tape and Labeled Diskettes

Appendix A – ASCII Code Set

Appendix B – Keyboard Summary

Appendix C – Submitting Batch Jobs in Stacked Format

Appendix D – VT100 Support

Index

Document Set

Tables

Figures

Chapter 1
Introducing the CLI

The Command Line Interpreter (CLI) is an interactive program that you use to communicate with the operating system. On most systems, the CLI begins running as soon as you log on. You can then use the CLI's command language to tell the system what you want it to do.

For example, by using CLI commands you can

- Create, delete, organize, copy, and move files, or display the contents of a file.
- Use peripheral devices, such as line printers and tape and diskette units.
- Execute programs and utilities.
- Obtain status information about the system, the processes (programs) that are running on it, and your own CLI working environment.

Your interaction with the CLI normally consists of a series of commands; as you enter each command, the CLI performs an action and/or displays information. The command line syntax (described in the next section), lets you easily build complex commands if you need to. You can enter more than one command on a single line, or cause a single command to repeat.

You can even build your own commands. You can define a series of CLI commands, called a *macro*. You can then execute these commands simply by typing the name of the file that contains the series of CLI commands. You'll learn about CLI macros in Chapter 4.

This chapter compares the two CLI programs you received with your system, and then tells how to format CLI commands and use control characters. The major sections proceed as follows.

- Two CLIs (CLI16 and CLI32)
- Getting Help
- Formatting a CLI Command Line
- How the CLI Processes Command Lines
- Using Comments in Your Macros
- Using Control Characters
- Positioning the Screen Cursor
- Locking the CLI

Two CLIs (CLI16 and CLI32)

Your operating system arrived with two CLIs: CLI16 (a 16–bit program) and CLI32 (a 32–bit program). The CLI32 offers features not found in CLI16, including the commands GROUPLIST and HISTORY, pseudomacros !INDEX and !SUBSTRING, and enhanced tools for macros. However, in most cases, the two CLIs behave exactly the same way.

The CLI that runs as your user process was selected by your system manager when he or she edited your user profile. In most cases, you can run either CLI under your initial CLI process by typing XEQ :CLI16 or XEQ :CLI32 and pressing NEW LINE.

You can tell which CLI is running by typing

```
) WRITE [!CLI] ↵
 CLIn
```

The response, CLI32 or CLI16, tells which CLI is running.

Getting Help

If you are in doubt about a CLI command, pseudomacro, system utility, or a related topic, you can ask the CLI to display an explanation of the item.

To list the topics for which help is available, give the command

```
) HELP ↵
```

To get help for one of the listed topics, type HELP followed by the topic name (which is preceded by an asterisk). For example, the command

```
) HELP *NEWLINE ↵
```

displays information about the NEW LINE key and its use. And the command

```
) HELP *COMMANDS ↵
```

displays a list of CLI commands for which individual help is available.

The individual descriptions of CLI commands and pseudomacros in Chapter 5 show the command you can use to obtain help information for that command or pseudomacro.

Chapter 5 also describes the /V switch for HELP. The command forms

```
HELP/V command–name
      and
HELPV command–name
```

display a verbose explanation of command–name.

Formatting a CLI Command Line

When writing a CLI command line, use this general format:

command*[/switch ...] [argument[/switch ...] ...]*

Every CLI command line must begin with either a CLI command (or a valid abbreviation of one) or a program or macro name. Depending on the command, macro, or program and your purpose for using it, the command line can also include *arguments* and/or *switches*. To enter the command line you have typed, press the NEW LINE key (represented by the symbol ⤶); the CLI will then try to execute the command. If what you typed is not a command, the CLI proceeds as explained later, in the section "How the CLI Processes Command Lines."

For realism in example command lines, this manual generally shows the CLI prompt). Do not type the prompt.

Arguments

An *argument* is a value that you supply to a command; the command then acts upon that data. For example, with the DELETE command, you supply one or more filenames; these arguments indicate which file(s) you want to delete.

Certain commands do not accept any arguments. Two of these are POP and SYSINFO.

Other commands accept arguments under certain circumstances. For example

) TIME ⤶ (Displays the current system time.)

) TIME 14:40⤶ (Sets the system time to 2:30 p.m.)

And still other commands *require* one or more arguments.

) DELETE MYFILE⤶ (Deletes a file called MYFILE.)

) RENAME FILE_IN FILE_OUT⤶ (Changes the name of file FILE_IN to FILE_OUT)

The format for each command (as shown in Chapter 5) indicates which arguments the command accepts or requires.

Switches

A *switch* is a qualifier that you append to a command (and sometimes an argument) in order to select a processing option. Every switch begins with a slash.

The QPRINT command, for example, which lets you print a file, accepts several switches. One of these, /NOTIFY, causes the system to send you a message when the printing operation is finished. So, if you want to print MYFILE and be notified when it is printed, use the command

) QPRINT/NOTIFY MYFILE ⤶

If you omit the /NOTIFY switch, the CLI prints the file without sending you any notification.

Some switches accept (or require) a value. To supply a value to a switch, append an equal sign (=) to the switch, followed by the value. The QPRINT command accepts the /COPIES= switch, which lets you specify the number of copies you want printed. This type of switch (called a *keyword switch*) requires you to supply a value. For example, to print three copies of MYFILE, you would use the command

```
) QPRINT/COPIES=3  MYFILE )
```

If you omit this switch, the CLI prints one copy of the file.

For information about the arguments and switches that pertain to a specific command, refer to the command dictionary in Chapter 5. The command descriptions show the syntax for each command, indicating the required or accepted arguments (if any). The description also lists the switches you can use.

Separators

If a command line contains one or more arguments, you must separate the command and each argument with a *separator*. (You do not use a separator before a switch.) The following characters or character combinations serve as separators in CLI command lines:

- One or more spaces (except at the beginning or end of a command line).
- One or more tabs (except at the beginning or end of a command line).
- A single comma.
- Any combination of spaces, tabs, and a single comma.

When it interprets a command line, the CLI converts each separator into a single comma. If the CLI displays the command line (after an error occurs, for example), a comma will appear where you used a separator.

For example, suppose you want to see the contents of files MINE1, MINE2, and MINE3. Here are several CLI commands that will do this for you. <TAB> means to press the TAB key once.

```
TYPE MINE1 MINE2 MINE3

TYPE,MINE1,MINE2,MINE3

TYPE MINE1 MINE2, MINE3

TYPE MINE1     MINE2     MINE3

TYPE MINE1<TAB>MINE2<TAB>     MINE3
```

The second line is an example of the *normalized form* of a CLI command line. That is, exactly one comma separates each command and argument. If, for example, you gave an erroneous command like this:

) 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.*

Using Comments in Your Macros

Comment lines are an important part of the CLI — particularly macros. Good comments in a macro can help anyone who uses the macro understand it, and if necessary update it for new conditions. There are three ways to write comments via the CLI: the COMMENT command, the lexical comment (CLI32 only), and the conditional operator comment. They work as follows.

COMMENT Command

A COMMENT command tells the CLI to ignore most characters up to the next NEW LINE or semicolon character. Lines beginning with COMMENT should not include special characters like semicolons (;), parentheses) or (, or brackets [, or], since the CLI will try to interpret these. (Rules appear later in this chapter.) These restrictions limit the use of COMMENT. A sample COMMENT line used in a macro is

```
COMMENT Sort filenames
FILESTATUS/SORT
```

Lexical Comment

In CLI32, two backslashes (\\) tell the CLI to ignore *all* characters (except a terminating ampersand line continuation character, &) up to the next NEW LINE. For clarity, you can (but need not) precede the backslashes with the command COMMENT. Lexical comments are much more flexible than COMMENT comments, but are available in CLI32 only. Sample lexical comment lines used in a macro are

```
\\ Sort filenames; also display time last modified.
FILESTATUS/SORT/TLM

COMMENT \\ Sort filenames; also display time last modified.
FILESTATUS/SORT/TLM
```

The ampersand line continuation character will work as usual to continue the line, regardless of the backslashes, if it precedes the delimiting NEW LINE character. However, if the ampersand immediately precedes the backslashes, the CLI will write it as usual; this provides an easy way to insert an ampersand character at the end of a line in a file you are creating with the CLI.

For example

```
) WRITE  Test1 // XXXX ↵
Test1

) WRITE Test2 // XXXX& ↵
&) ↵
Test2

) WRITE Test3& // ↵
Test3&
```

Conditional Operator Comment

A conditional operator comment consists of a conditional operator that will never return True, followed by text and an [!END] terminator. The text can include any characters and span multiple lines. This works in both CLIs. Conditional operator comments are less clear than the COMMENT command (because they begin with a conditional operator), but far more flexible (they can include any characters and span multiple lines). They are most useful for multiple–line comments. A sample conditional operator comment used in a macro is

```
[!EQUAL,COMMENT,C]Sort filenames; also display
time last modified.[!END]
FILESTATUS/SORT/TLM
```

Choose from these comment techniques according to your needs and the CLI to which the comment indicator will be issued.

Using Control Characters

Normally you use the CTRL (control) or CMD (command) key, like the SHIFT key; that is, you press and hold it while you type a character key. And like the SHIFT key, CTRL gives yet another meaning to the character key that you press with it. We call this key combination a *control character.*

When referring to a control character, we use the following notation:

CTRL–character

For example, CTRL–C means press the CTRL key and the letter C at the same time. (There is no difference between upper– and lowercase letters when used with the CTRL key).

A control character, rather than producing a symbol, generates a code that instructs the system or your terminal to perform a special function. For example, you can use control characters to freeze text on your screen or to interrupt a program that you are running. The CLI accepts and executes control characters even while it is executing some other CLI command.

Table 1–1 summarizes the control characters.

Table 1–1 Control Characters and Special Keys

Key(s)	What It Does
CTRL–O	Discards display for the portion of the command that remains to be executed, or until you type CTRL–O again, whichever happens first. CTRL–O does not halt the program. Some programs may execute faster if you type CTRL–O because writing data to the terminal is a relatively slow operation.
CTRL–S	Suspends display. Display resumes where it stopped when you press CTRL–Q. CTRL–S and CTRL–Q can help you read long files on a CRT, when display is too fast to read. Or you can press the HOLD key (if present) to stop and start display.
CTRL–Q	Resumes display. If you stopped display with CTRL–S, use this. If you stopped display with CTRL–O, CTRL–Q has no effect.
CTRL–U	Erases the current input line. This is handy when you have typed a long, erroneous command line and don't want to press the DEL key many times to erase it.
CTRL–P	Accept the next character literally, without interpretation. Use this to input a control character without having the system act on it.
CTRL–C CTRL–A	Interrupts execution of a CLI command. You'll find this sequence useful. For example, if you are using the CLI to display the contents of a file, CTRL–C CTRL–A can stop the display before the CLI reaches the end of the file. Or you can use this sequence to cancel a CLI command line you are typing.
CTRL–C CTRL–B	Aborts the process that issues it (like the CLI or a text editor). Avoid using this unless you really want to abort the process.
CTRL–D CTRL–D	In AOS/VS II, signals an end of file — which usually aborts the issuing process. Generally, avoid this sequence.
CTRL–C CTRL–E	In AOS/VS II, creates a memory–image break file (useful for debugging), and aborts the issuing process. Generally, avoid this.
DEL key	Erases the last character typed. On a hardcopy terminal, DEL echoes as _ (underscore) for each character erased.
BREAK key	Enters the break sequence. On newer CRTs, press the CMD key and, while holding it down, press the BREAK/ESC key; on DASHER D2 CRTs, press the BREAK key; on hardcopy terminals, press the BRK key. The break sequence is useful when you have accidentally typed a binary file and the terminal does not respond to CTRL–C CTRL–A. Do not type the break sequence on the system console unless you want to enter the SCP CLI (explained in *Managing AOS/VS and AOS/VS II*).

Screenedit Control Characters

The Screenedit control characters, like the left– and rightarrow keys, help you move the cursor around the terminal screen. They also help you to insert characters, insert tabs, and delete characters. To use the Screenedit characters, you must turn Screenedit mode on. (For more information, see the description of the SCREENEDIT command in Chapter 5.)

Table 1–2 summarizes the Screenedit control characters.

Table 1–2 Screenedit Control Characters

Character	What It Does
CTRL–B	Moves the cursor to the end of the previous word.
CTRL–E	Toggles insertion mode. The first CTRL–E begins insertion mode. You can then type character(s), inserting them at the current position. Type a second CTRL–E to end insertion mode.
CTRL–F	Moves the cursor to the beginning of the next word.
CTRL–H	Moves the cursor to the beginning of the current line (equivalent to pressing the HOME key).
CTRL–I	Inserts a tab (equivalent to pressing the TAB key).
CTRL–K	Erases everything from the cursor position to the right (equivalent to pressing the ERASE EOL key).
CTRL–L	Enters the current line and clears the screen (equivalent to pressing the ERASE PAGE key).
CTRL–X	Moves the cursor one character to the right (equivalent to pressing the rightarrow key, !).
CTRL–Y	Moves the cursor one character to the left (equivalent to pressing the leftarrow key, z).

The following dialog shows how the Screenedit control characters let you change a command easily *without* re–entering it character by character. Remember that control characters are case–insensitive so that, for example, CTRL–A and CTRL–a have identical results. Better yet, try the four cases on your terminal.

The following dialog shows how to repair a command.

```
) TYPO MYFILE1 MYFILE2 MYFILE3 ↵
Error: Not a command, macro or program, TYPO,
TYPO MYFILE1,MYFILE2,MYFILE3
)
```

Press CTRL–A to redisplay the command.
Press CTRL–H (or the HOME key) to move the cursor to the beginning of the incorrect command.

Press CTRL–F to move the cursor forward to the M of MYFILE1.
Press CTRL–B to move the cursor backward to the O of TYPO.
Type E to correct the incorrect word.

Press NEW LINE to give the corrected command to the CLI. If files MYFILE1, MYFILE2, and MYFILE3 exist, you will now see their contents. Otherwise, you see up to three warning messages about a nonexistent file.

The following dialog shows how to give three similar commands with a minimum of keystrokes.

```
) WRITE Step 1 is to open the file. ↵
Step 1 is to open the file.
)
```

Press CTRL–A to redisplay the command.
Press CTRL–H to move the cursor to the beginning of the command.
Press CTRL–F twice to move the cursor to the 1.
Type 2 to correct the incorrect digit.
Press CTRL–F three times to move the cursor to the beginning of the word open.
Type the following characters: look at the file.
Step 2 is to look at the file.

Press CTRL–A to redisplay the command as it appears on the previous line.
Press CTRL–H to move the cursor to the beginning of the command.
Press CTRL–F twice to move the cursor to the 2.
Type 3 to correct the incorrect digit.
Press CTRL–F three times to move the cursor to the beginning of the word look.
Type the following characters: close the file
Press CTRL–K or the ERASE EOL key (either one erases characters to the right of the cursor)
Step 3 is to close the file.

The following dialog shows how to insert characters in a command line.

) WRITE Now is the time demonstrate insertion mode. ↵

Now is the time demonstrate insertion mode.

Press CTRL–A to redisplay the command.
Press CTRL–H to move the cursor to the beginning of the command.
Press CTRL–F five times to move the cursor on the "d" of "demonstrate."
Press CTRL–E to begin insertion mode.
Type t and o, and press the space bar.
Press CTRL–E to end insertion mode.
Press CTRL–A to move the cursor to the end of the current line.
Press NEW LINE to give the command to the CLI.
Now is the time to demonstrate insertion mode.

The following dialog shows how to preview a command line, change it, and issue the command.

) WRITE TIME; WRITE No miszteaks here. ↵

TIME
No miszteaks here.

Press CTRL–A to redisplay the command.
Press CTRL–B three times to move the cursor on the "o" for "No."
Press CTRL–F to move the cursor on the "m" of "miszteaks."
Press the rightarrow key (or CTRL–X) four times to move the cursor on "t."
Press the DEL key to delete the "z;" it remains on "t."
Press the rightarrow key (or CTRL–X) once to move the cursor on the "e."

Type a, k, and e to overwrite "eak." The line now appears as
WRITE TIME; WRITE No mistakes here. (The cursor is at the second "s" of "mistakes."

Press CTRL–H to move the cursor to the beginning of the line.
Press the space bar five times to remove the first word. The line is now correct.
Press the NEW LINE key. This turns the command line over to the CLI. The CLI responds with the following two lines.

h:m:s (The current system time)
No mistakes here.
)

As you can see, the Screenedit control characters are quite useful as you repair incorrect commands, give successive similar commands, and preview commands before you actually give them. In particular, CTRL–A saves you much time since it retrieves the previous command. The HISTORY command that Chapter 5 describes lets you retrieve any one of the previous 25 lines that you gave to the CLI.

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.CLI                                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

Chapter 2
Using the File System

This chapter describes the AOS/VS and AOS/VS II file system, and explains how to use the CLI to create, organize, protect, and refer to files. This information appears in the following major sections:

- Files
- Directory Files
- Pathnames
- Link Files
- Access Control
- User Groups
- File Permanence
- Generic Files
- Device and Queue Names
- File Types

Files

A *file* is a named collection of information, such as a list of telephone numbers, a memo, or a COBOL program. A file can reside on disk, magnetic tape, or diskette. In this manual, when we refer to a file, we normally mean a disk file, unless specified otherwise.

Every file has a unique name that lets you refer to the file individually. A filename consists of from 1 through 31 characters. When naming a file, you can use these characters.

A–Z, a–z	Alphabetic characters (the system converts lowercase to uppercase)
0–9	Numerals
_	Underscore
$	Dollar sign
?	Question mark
.	Period

The AOS/VS and AOS/VS II file systems and the CLI are *case–insensitive* — that is, they do not distinguish between uppercase and lowercase alphabetic characters in filenames. The operating system converts all lowercase characters to uppercase before storing them. Therefore, to the CLI and operating systems, the filenames FILE1, File1, and file1 all refer to the same file.

Filename Suffixes

To help identify the information a file contains, the system and some of its utilities use a filename *suffix*, also known as an *extension*. A suffix begins with a period, and is followed by characters (usually two or three) that describe file contents.

A common filename suffix is .PR, which identifies a program file; the Link utility's program file is called LINK.PR. Another important filename suffix is .CLI, which you use with a CLI macro file (as described in Chapter 4).

Table 2–1 lists many of the suffixes that the system and its utilities recognize. If you write programs, we encourage you to use the standard suffixes for your source files.

Table 2–1 Sample Filename Suffixes

Suffix	Meaning	Example
.BRK	Break file (created by terminating program)	MY_PROG.BRK
.C	C language source file	MYPROG.C
.CLI	CLI macro file	UP.CLI
.COB	COBOL source file	PAYROLL_010.COB
.CONFIG	VSGEN configuration file (created by AOS/VS II VSGEN)	IMG017.CONFIG
.CSF	VSGEN customer specification file (created by AOS/VS VSGEN)	IMG017.CSF
.DG	DG/L source file	SYS_008.DG
.ED	SED text editor edit status file	MYPROG.ED
.FR	FORTRAN IV or FORTRAN 5 source file	MYPROG.FR
.F77	FORTRAN 77 source file	MYPROG.F77
.JOB	Batch input file (created by CLI in initial user directory; deleted when job completes)	?113.CLI.00002.JOB
.LB	Unshared library	LIB05.LB
.OB	Object file (created by compiler/assembler)	MYPROG.OB
.PAS	Pascal source file	MYPROG.PAS
.PL1	PL/1 source file	MYPROG.PL1
.PR	Program (executable) file (created by Link program)	MYPROG.PR
.RG	RPG II source file	MYPROG.RG
.SC	Text editor store–changes file (editor creates, then deletes at normal termination)	MYPROG.SC
.SR	Assembly language source file	MYPROG.SR
.ST	Symbol table file (created by Link program)	MYPROG.ST
.TCS	Text Control System (TCS) master file	TEXT_03.TCS
.TMP	Temporary file (created by utility and deleted at normal termination)	?SORT_003.TMP

In certain circumstances, you do not need to include the suffix when you refer to a file. The system can often determine the file you want to use by context.

For example, if you use the XEQ command to execute a program, the system knows that the argument refers to a program file. In this case, you do not have to include the .PR suffix because the system assumes that the file has that suffix.

To run the SED text editor (whose filename is SED.PR), you can issue the command

```
) XEQ SE ↵
```

The system can still locate a file that does not have the expected filename suffix. After not finding the file with the expected suffix, the system looks for the file exactly as you specified it.

So, you can execute a program file called MYPROGRAM (without the .PR suffix) with the command

```
) XEQ MYPROGRAM ↵
```

The system first looks for MYPROGRAM.PR; because that file does not exist, the system next looks for a file called MYPROGRAM.

When you specify only a pathname, the CLI searches for the pathname with suffix .CLI; if it cannot find pathname.CLI, then it searches for the pathname alone. For example, if you type MYFILE and press NEW LINE, the CLI looks for MYFILE.CLI, and then for MYFILE.

CLI32 takes an additional step: if it cannot find the pathname, it searches for pathname with a .PR suffix; if CLI32 finds that file, it tries to execute it (as with XEQ pathname). So you can execute a program from CLI32 by typing the program name only, without an XEQ, EXECUTE, or PROCESS command.

Unless you want to execute a program or run a macro, you must include the suffix. An example is the MOVE command to make a copy of a file. If you give a MOVE command that includes MYPROGRAM and file MYPROGRAM doesn't exist (even if file MYPROGRAM.PR or MYPROGRAM.CLI does exist), the CLI responds with an error message about a nonexistent file.

If in doubt, include the filename suffix to specify a file.

Directory Files

A *directory* is a file that contains information about other files. You can think of a directory as a place to store files. Directories help you organize your files. You might want to use a directory to store program files, files pertaining to a particular project, or memos and correspondence; the organization is up to you.

Any directory can contain one or more other directories, which are called *subdirectories*. This lets you build a directory structure with multiple levels, so that you can group your files in whatever way suits your needs.

Control Point Directories

There is a special type of directory file called a *control point directory* (CPD). Unlike a standard directory, a control point directory has a specific size limit. You specify the control point directory's maximum size when you create it, and you can change the maximum size as necessary.

By using control point directories, you can better regulate the use of disk space.

The System Directory Tree

All files and directories on your system are part of a single directory structure. This structure is like an inverted tree. It begins at the *root directory* (represented by a colon, :) and branches out to provide an overall organization according to the needs of the system. Within this hierarchy, any one directory is *superior* to the directories below it, and *subordinate* to directories above it.

Figure 2–1 illustrates a directory tree. The root directory (:) is the highest directory on the system. Subordinate to the root directory are three other directories: PER, UTIL, and UDD.

- PER (the peripherals directory) contains generic files and entries for peripheral devices. (This directory and its contents are described later in this chapter.)
- UTIL (the utilities directory) contains system utility programs.
- UDD (the user directory directory) contains a directory for each person who has a user profile on the system.

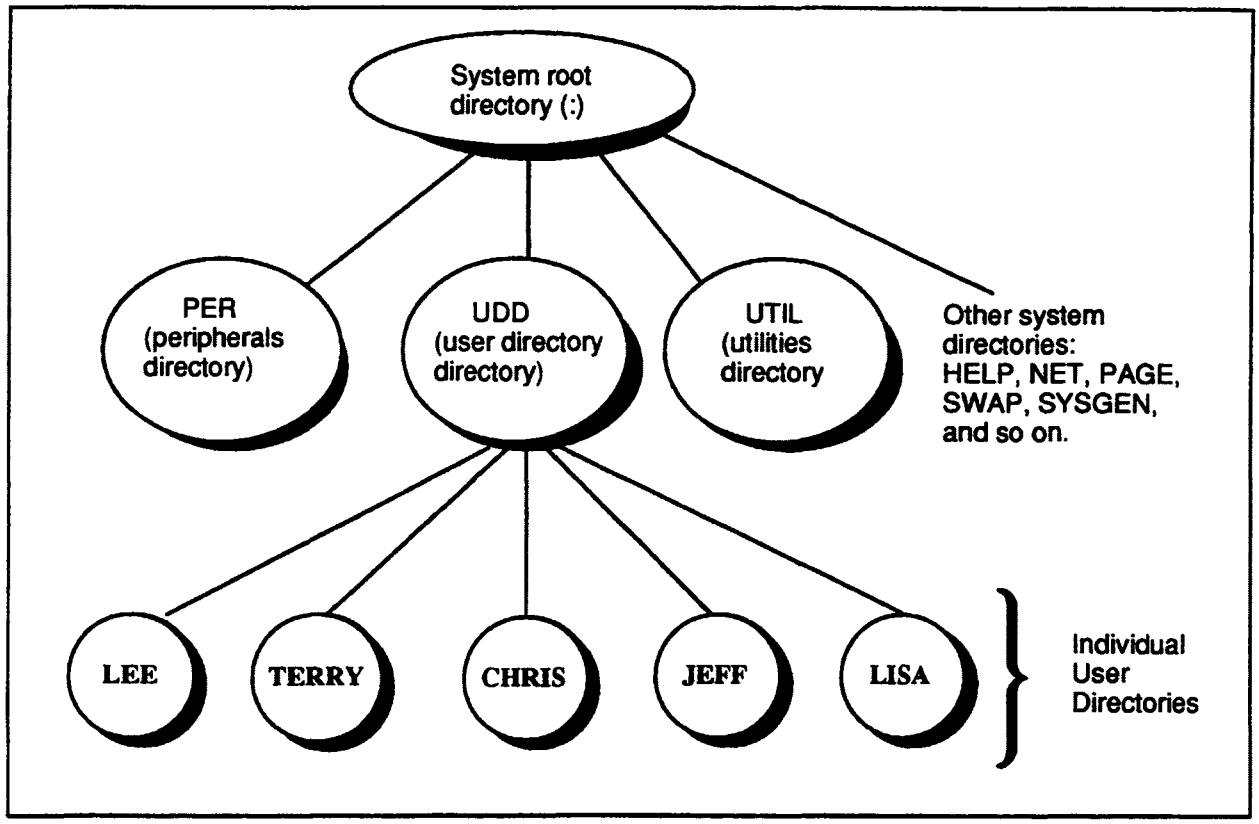


Figure 2–1 A Directory Tree

User Directories

Everyone who has a profile on the system is given a directory that is subordinate to the UDD directory. The user directory has the same name as its owner's username. For example, user LEE has a directory called LEE under UDD.

This directory, called your *initial user* directory, is the starting point for your own directory tree. You organize your files and subdirectories within this directory structure. You can work with files and directories outside your directory tree only if users grant you access privileges (as described later in this chapter). In the same way, your directory tree can prevent unauthorized access by others.

To create a directory, you use the CREATE command with the /DIRECTORY switch. For example, to create directory MEMOS, you would enter the command:

```
) CREATE/DIRECTORY MEMOS ↵
```

Then you could use the FILESTATUS command with the /ASSORTMENT switch, to display information about the new directory.

```
) FILESTATUS/ASSORTMENT MEMOS ↵
Directory xxx          (xxx is the pathname of the working directory)
MEMOS                  DIR  14-Nov-90  14:42:42        0
```

In the system response, *DIR* tells you that the file is a directory file. The other items report the date and time the file was created, and the file's length (in bytes.)

A standard directory, created with the CREATE/DIR command, is flexible; it can grow as needed as you add files to it. A control point directory has a maximum size, which the system will enforce. All user directories are control point directories, which means that the space within them (including all space within standard directories in them) is limited. To learn the amount of space available in your user directory, use the SPACE command:

```
) DIR/I ↵                (Move to initial user directory)
) SPACE ↵                (Use the SPACE command)

 Max   20000, Cur   322, Rem   19678
```

The CLI shows the maximum number of 512–byte blocks available *(Max)*, the number of those blocks currently used *(Cur)*, and the number remaining *(Rem)*.

To create a control point directory, use the /MAXSIZE= switch with (or instead of) the /DIRECTORY switch. Specify the number of blocks (1 block equals 512 bytes) you will allow for the contents of this directory. For example,

```
) CREATE/MAXSIZE=10000 REPORTS ↵
) FILESTATUS/ASSORTMENT REPORTS ↵

 Directory xxx
 REPORTS          CP        14–Nov–90 14:52:42   0
```

With AOS/VS or with CLI16, the SPACE command cannot display disk usage information on a standard directory; if you try, the CLI will return the error message *Not a control point directory*. Therefore, if you are using AOS/VS or CLI16 and space accounting is important, a control point directory has a major advantage over a standard directory. With AOS/VS II and CLI32, the SPACE command *does* return the amount of space used in a standard directory; thus with CLI32 you can use a standard directory even if space accounting is important. The following example, under AOS/VS II, compares the behavior of the two CLIs.

```
) WRITE [!CLI] ↵
 CLI16                          (Running CLI16)

) CREATE/DIR MYDIR ↵
) CREATE MYDIR:XFILE ↵
) SPACE MYDIR ↵

 Warning: File is not a control point directory, File MYDIR   (Error message)

) XEQ :CLI32 ↵                  (Run CLI32)
 AOS/VS II CLI32...             (CLI32 displays program banner)

) SPACE MYDIR ↵
 Max   NA, Cur  1, Rem   NA     (CLI32 reports that 1 block is used)
```

If, after creating and using a directory, you decide that the other type would be more suitable, create a directory of the desired type, move all files from the original directory to it, and the delete the original directory. For example,

```
) CREATE/DIR  MEMOS.1 ⤶
) DIRECTORY  MEMOS ⤶
) MOVE/V  ^MEMOS.1 ⤶
... (System verifies files copied) ...
) DIRECTORY  ^ ⤶
) DELETE/V  MEMOS:# ⤶
... (System verifies files deleted) ...
) RENAME  MEMOS.1  MEMOS ⤶
```

The ^ and # characters are a pathname prefix and template respectively; they are explained in following sections.

Working Directory

Normally when you log on to the system, your initial directory becomes your *working directory*. This is your current position within the directory tree.

The files in your working directory are immediately available to you. To refer to any file in your working directory, all you need is its filename. For example, to delete FILE1 from your working directory you would use the command

```
) DELETE FILE1 ⤶
```

Because FILE1 is in the working directory, the CLI immediately finds the file and deletes it.

If, on the other hand, you wanted to delete a file that isn't in your working directory, you must tell the CLI where it can find the file. To do this, you must furnish a *pathname* to the file.

Pathnames

A *pathname* lets you refer to a file outside your working directory. You can use a pathname when you do not want to move to a different directory. In effect, a pathname is a map that lets the system find a file anywhere within the directory tree.

Pathname Prefixes

You begin a pathname at one of the following locations within the system directory tree. For each of these locations, the CLI recognizes a special symbol, called a *prefix*, which you use to begin the pathname. The directories specified by these symbols are

- The working directory (=)
- The root directory (:)
- The peripheral directory, PER (@)
- The immediately superior directory (^)

To refer to any of these directories, you can use the prefix alone. For example, you can use these prefixes with the DIRECTORY command, which sets your working directory and allows you to move from one directory to another.

The command

```
) DIRECTORY ^
```

moves you to the immediately superior directory. To move to the peripherals directory (PER), you could use the command

```
) DIRECTORY @
```

The prefix for your working directory, = , is normally optional because the system automatically looks in your working directory if you do not specify another prefix. You can use this prefix, however, when you want the pathname to refer explicitly to a file in your working directory. This will prevent the CLI from looking elsewhere for a file if it does not reside in the working directory. Table 2–2 explains the pathname prefixes.

Table 2–2 Pathname Prefixes and Meanings

Prefix	Meaning
:	Begin at the root directory.
=	Begin at the working directory. You can usually omit this prefix because the system automatically searches your working directory if you do not specify a prefix. You can use this prefix, however, to force the CLI to look *only* in your working directory and not scan the directories on your search list.
^	Begin at the immediately superior (parent) directory. The caret prefix frees you from having to specify the superior directory by name. You can use more than one caret to move up more than one directory. For example, ^^ moves you up two superior directories to your grandparent directory.
@	Begin at the peripheral directory, :PER. This prefix is useful for referring to the filename for a device (such as a terminal or printer) or a generic file.

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.)

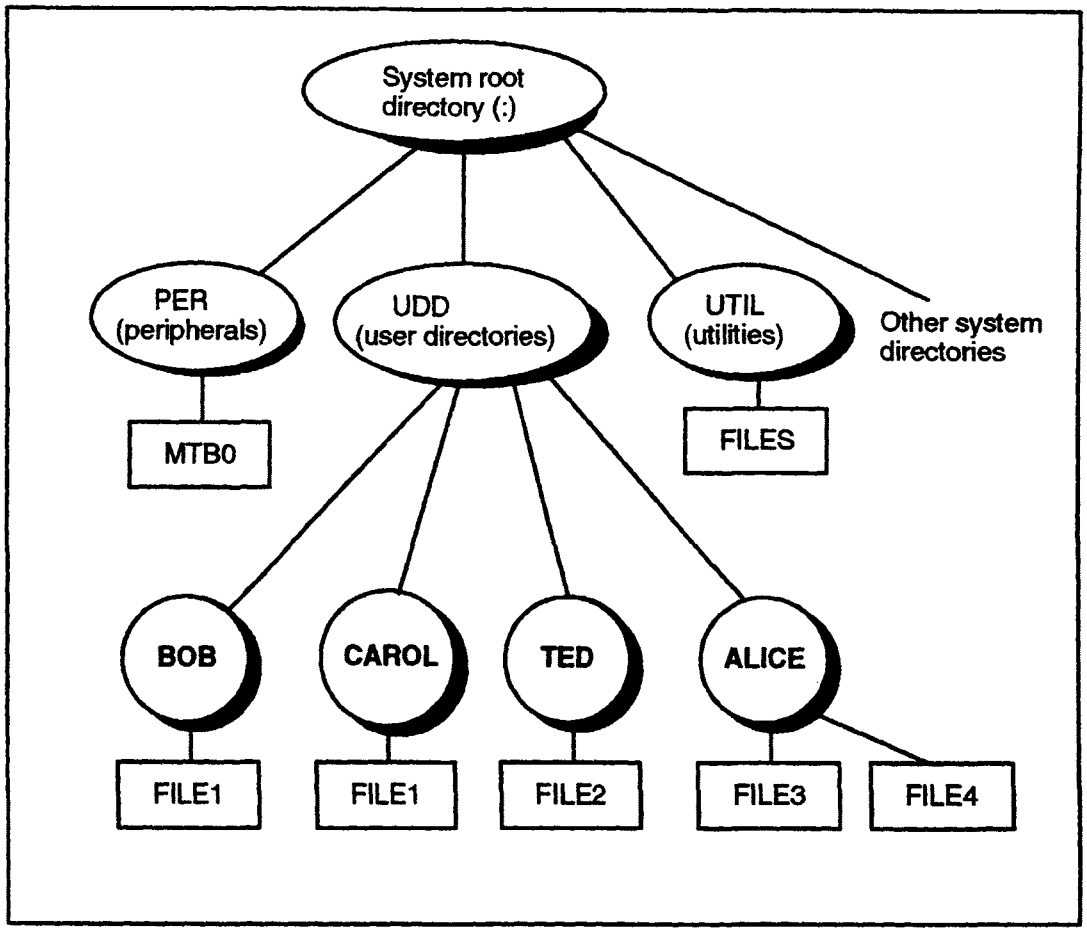


Figure 2–2 A Directory Tree with User Files

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).

Table 2–3 Pathnames from the Directory CAROL in Figure 2–2

To refer to	You can use the pathnames
FILE1 (under CAROL)	FILE1, =FILE1, or :UDD:CAROL:FILE1
FILE1 (under BOB)	:UDD:BOB:FILE1, or ^BOB:FILE1
MTB0	:PER:MTB0, ^^PER:MTB0, or @MTB0
FILE5	:UTIL:FILE5 or ^^UTIL:FILE5
TED	:UDD:TE or ^TED
CAROL	= or :UDD:CAROL
:	: or ^^

Parts of a Pathname

The CLI provides several ways for you to refer to various components of a pathname. They are based on a software construct called a *pseudomacro.*

A pseudomacro accepts zero or more arguments and returns exactly one result. For example, !DATE is a pseudomacro that accepts zero arguments and returns the date from the system. A command that invokes this pseudomacro is

```
) WRITE [!DATE]
```

The following pseudomacros return various parts of a pathname.

- !EDIRECTORY
- !EEXTENSION
- !EFILENAME
- !ENAME
- !EPREFIX

Figure 2–3 illustrates what each of these pseudomacros represents. When you use a pseudomacro within a CLI command, the CLI replaces the pseudomacro with the value it represents. For more information about these pseudomacros, see Chapter 5.

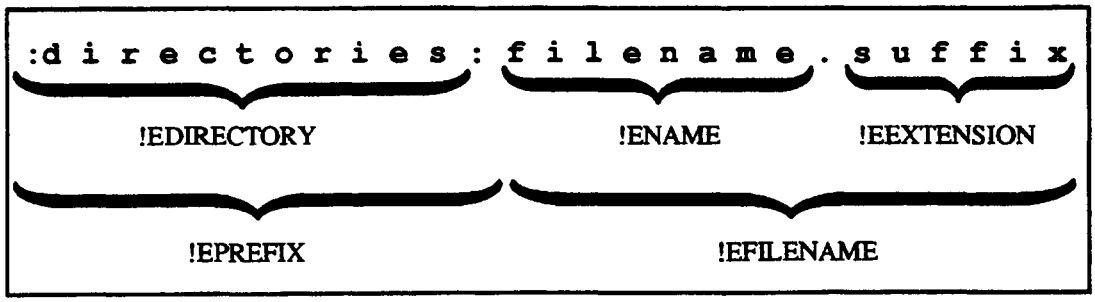


Figure 2–3 The Components of a Pathname

For example, suppose the pathname of a file you are interested in is:

:UDD:JEFF:HOMEWORK:MATH:FIGURE_23.VERSION1

Here are examples of the five pseudomacros and the values they return.

[!EDIRECTORY :UDD:JEFF:HOMEWORK:MATH:FIGURE_23.VERSION1]

:UDD:JEFF:HOMEWORK:MATH

[!EEXTENSION :UDD:JEFF:HOMEWORK:MATH:FIGURE_23.VERSION1]

.VERSION1

[!EFILENAME :UDD:JEFF:HOMEWORK:MATH:FIGURE_23.VERSION1]

FIGURE_23.VERSION1

[!ENAME :UDD:JEFF:HOMEWORK:MATH:FIGURE_23.VERSION1]

FIGURE_23

[!EPREFIX :UDD:JEFF:HOMEWORK:MATH:FIGURE_23.VERSION1]

:UDD:JEFF:HOMEWORK:MATH:

Template Characters

A *template* character (also called a *wildcard*) is a special character that the CLI interprets symbolically rather than literally. Templates are valuable tools when you forget the name of a file, or you want to refer to a group of files with a single name.

A pathname template is a pathname that contains template characters. In a sense, pathname prefixes are template characters since they are also symbolic characters. You can, however, use prefixes only at the beginning of a pathname. The template characters that we describe in this section can appear anywhere within the pathname.

Table 2–4 lists the five CLI template characters and the type of filenames that they match. The next sections describe each template character and give examples of its use.

Table 2–4 Template Characters and Meanings

Character	Meaning
*	(asterisk) Matches any single character except a period. For example, the template TEST* matches TESTA, TEST1, and all other five–character names beginning with the characters TEST (except TEST followed by a period).
–	(hyphen) Matches any character string that does not contain a period (including a null string). For example, the template TEST– matches TEST, TEST1, TEST89, TEST123, TESTING, and any other name that begins with the characters TEST and does not contain any periods. The template would *not* match TEST.PR.
+	(plus sign) Matches any character string (including a null string). For example, the template T+ST matches TST, T.ST, TEST, TOAST, and any other name that begins with the character T and ends with the characters ST.
#	(number sign) Expands to +, and includes all subordinate directories and their contents. For example, the template :UDD:CAROL:TEST:# represents all files and directories subordinate to :UDD:CAROL:TEST (if any).
\	(backslash) Excludes the filename that follows. The filename can be a directory name and can include template characters. The exclusion ends with the next backslash or colon (:). For example, the template TEST+\TEST.DIR represents all files that begin with the characters TEST, except file TEST.DIR. This template is often used with # (for example, FILESTATUS/AS #\TEST.DIR).

You can think of a pathname or a filename that contains a template character as a filter that allows certain filenames to pass through. For example, consider the following filenames.

FILE1 FILE2 FILE34 MY_FILE PROG999 PROG.SR PROG.TEST

The following filenames with templates filter or allow through (that is, match) the corresponding filenames.

Filename Template	Filenames Matched
FILE*	FILE1, FILE2
FILE–	FILE1, FILE2, FILE34
PROG.–	PROG.SR, PROG.TEST
PROG+	PROG.SR, PROG999,PROG.TEST
+	All filenames
#	All filenames plus all subordinate directories and their contents
+\P+	FILE1, FILE2, FILE34, MY_FILE
+\P+\F+	MY_FILE
FILE1	(no template) FILE1
JEFF+	None

You can use the FILESTATUS command (or a WRITE command and !FILENAMES pseudomacro) to list the names of selected files in a directory. For example, the command

```
) FILESTATUS  TEMP+ SPECIAL+ ↵
```

lists the names of all files in your working directory that begin with TEMP or SPECIAL. This is quite useful in previewing the effects of commands that accept filenames with templates. For example, suppose you want to know *in advance* the names of the files that the command

```
) DELETE TEMP_AUG+
```

would delete. Type

```
) FILESTATUS  TEMP_AUG+ ↵
```

to see a listing of the filenames.

Asterisk (*)

An asterisk, when used in a filename, represents any single character other than a period.

If you use this template with the following FILESTATUS command, the CLI lists all files in your working directory that have a single character name.

```
) FILESTATUS  * ↵
```

Suppose, for example, that you have a file called TEST and several versions of it, which you have named TEST1, TEST2, and so on. If you want to display a list of these files, you can use the command

```
) FILESTATUS  TEST* ↵
```

The CLI responds by listing TEST, TEST1, TEST2, and all other files that begin with the letters TEST and have exactly five characters.

This template would not, however, match TEST10 or TEST20. To locate these files you could use the command

```
) FILESTATUS TEST** ↵
```

Hyphen (–)

The hyphen, when used in a filename, represents any character string that does not contain a period, even a null string.

For example, the command

```
FILESTATUS TEST– ↵
```

lists *all* files in the working directory whose names begin with TEST, except those that contain a period. In other words, the template matches TEST, TEST1, TEST99, TESTING, and TEST_PROGRAMS, but not TEST.DIR or TEST.PR.

Plus Sign (+)

The plus sign, when used in a filename, matches *any* character string. This template character differs from the hyphen in that it also matches a string that includes one or more periods.

When used alone, this template matches *every* file in the working directory. The following command, therefore, is one you should use with care:

```
DELETE +
```

If you want to list all files in the working directory whose names include the letters TEST, you could use the command

```
) FILESTATUS +TEST+↵
```

This template matches names such as TEST, FIRST_TEST, TESTING, AB_TEST_CODE, TEST.DIR, and TEST.PR.

This template character is also useful when you want to work with files that have the same name, but different filename suffixes. The next command moves a copy of the files PROG1.PR, PROG1.OB, PROG1.LS, and PROG1.F77 from the working directory to a subdirectory called MYPROGRAM.

```
) MOVE MYPROGRAM PROG1+↵
```

Number Sign (#)

Unlike other template characters that you can use as part of a filename, the number sign represents a directory, and matches the contents of that directory and all subordinate files.

For example, if you use this template alone, it matches the contents of the working directory and of all subdirectories. In this case, the template is equivalent to

```
=+:+:+
```

(and so on through the bottom of the directory tree).

If your working directory is your initial directory, you can use this template character to obtain a listing of your entire directory tree.

```
) DIRECTORY/I ↵
) FILESTATUS/ASSORTMENT # ↵
```

The first of these two commands moves you to your initial directory. The FILESTATUS command lists the contents of that directory and all its subdirectories (and all directories subordinate to the subdirectories).

If you use it as part of a pathname, this template character expands to the filename immediately preceding the template, and all subordinate files (if any).

```
) FILESTATUS  TESTING:# ↵
```

This pathname matches the file TESTING (in the working directory); if TESTING is a directory, the template also matches the contents of the directory and all its subordinate files.

You can use this template to search subdirectories for a specific file. For example, to search all subdirectories for a file (or files) called LOST, you could use the command

```
) FILESTATUS  #:LOST ↵
```

The next command lists all program files (files with the .PR suffix) contained in the subdirectory TESTING or any of its subdirectories.

```
) FILESTATUS  TESTING:#:+.PR ↵
```

The next commands delete from your entire directory tree all files that have the filename suffix .OLD.

```
) DIRECTORY/I ↵
) DELETE/V  #:+.OL    ↵
.
(CLI lists files deleted)
.
)
```

Backslash (\)

You use the backslash to specify one or more files that you do not want to use even if they match the preceding template. The backslash means *except the following file(s).*

For example, the template

FILE+\FILE1

matches all filenames beginning with the letters "FILE", except FILE1.

You can use template characters after the backslash. The next template excludes all files that end with the .PR filename suffix.

FILE+\+.PR

Thus, this template matches FILE, FILE1, FILE.TEST, but not FILE.PR or FILE1.PR.

You can also use more than one backslash within a template. For example

+\FILE2\+.SR

matches every filename in the working directory except FILE2 and files that end with the .SR filename suffix.

You cannot include a colon within a string of text to be excluded. A colon effectively ends the string that is to be excluded. For example, the template

FILE+\FILE1:A

excludes FILE1:A, but it matches FILE:A, FILE2:A, and so on. In other words, the CLI applies \FILE1 to FILE+; the resulting file(s) are then used with :A.

Search Lists

Quite often you'll want to work with files that are in different directories. You might, for example, want to edit a text file in your working directory by running a text editor program that is in the system's :UTIL directory. Although you can use a pathname to refer to a file in another directory, the CLI provides a *search list* feature that can help eliminate the need for entering long pathnames. Your search list is a list of directories that the system automatically searches when it can't find a file in your working directory.

In other words, if you refer to FILE1, for example, but there is no FILE1 in your working directory, the system begins looking through each directory on your search list until it finds FILE1. (If FILE1 does not exist in any of these directories, the CLI then displays an error message.)

The CLI's SEARCHLIST command lets you display your current search list or change it. (See the description of this command in Chapter 5.)

You should use your search list to include the directories that contain files you frequently use. The seach list allows you to refer to these files without using a pathname.

You can place as many as eight directories on your search list. The system scans the directories in the order you list them, so place the directories that you use more often toward the beginning of the list.

You can use your search list to determine which of similarly named files the system will use. Suppose, for example, that your search list is

:UDD:LEE :UTIL :UDD:COMMON

The :UTIL directory contains a program called UPDATE.PR, but you have just created a newer version of the program and given it the same name. Your program resides in the directory :UDD:LEE:MYPROGRAMS. To ensure that the system uses your program rather than the one in :UTIL, place the directory :UDD:LEE:MYPROGRAMS on your search list so that it appears *before* :UTIL. For example,

:UDD:LEE :UDD:LEE:MYPROGRAMS :UTIL :UDD:COMMON

As mentioned earlier, you can use the = prefix with a pathname to specify a file in your working directory. This prefix explicitly refers to your working directory, so the CLI looks in that directory only; the CLI does *not* use your search list in this case.

NOTE: The CLI will *not* use your search list with certain commands. The DELETE and DIRECTORY commands, for example, will not look outside your working directory if you do not use a pathname prefix.

Link Files

Link files offer another way to access a file. A link is a file that points to another file; the link file contains a pathname. The file pointed to is called the *resolution file*.

For example, you can create a link file called MYPROG.PR associated it with the resolution file :UDD:SALLY:TEST_PROGRAMS:MYPROG.PR. To do so, type

```
) CREATE/LINK  MYPROG.PR   :UDD:SALLY:TEST_PROGRAMS:MYPROG.PR ↵
```

Now, to execute MYPROG.PR, you can use the link, MYPROG.PR, rather than the long pathname. The system recognizes that MYPROG.PR is a link, and uses the pathname in it. The command

```
) XEQ  MYPROG.PR ↵
```

is equivalent to

```
) XEQ  :UDD:SALLY:TEST_PROGRAMS:MYPROG.PR ↵
```

If you delete the link (as with DELETE MYPROG.PR ↵), the system deletes the link only; the resolultion file (:UDD:SALLY:TEST_PROGRAMS:MYPROG.PR) remains.

You can also use a link as part of a larger pathname. Assume the directory :UDD:SALLY:REPORTS contains monthly progress reports. You can use the following command to create a link to that directory.

```
) CREATE/LINK  REPORTS  :UDD:SALLY:REPORTS ↵
```

You can use this link within a pathname. For example, to type the contents of a report in the directory, you could use the command

```
) TYPE  REPORTS:NOV ↵
```

The system displays the contents of the file :UDD:SALLY:REPORTS:NOV. There is a disadvantage to using links within pathnames this way: the FILESTATUS command will not resolve the link, therefore you cannot rely on FILESTATUS to tell you whether the resolution file exists. For example, the command FILESTATUS REPORTS:NOV would not show the existence of file NOV.

If you use a link *within* a pathname, the part of the pathname that precedes the link must specify the path to the link. In other words, the CLI treats everything up to and including the link filename as the link itself. For example

```
) FILES/ASSORT  MYDIR:CORRESP:MEMOS ↵
```

Directory :UDD:JAN:MYDIR:CORRESP

MEMOS *DIR* *14-Nov-90* (MEMOS is a directory.)

```
) CREATE/LINK  MEMOS  CORRESP:MEMOS ↵
) DIRECTORY  ^ ↵
) TYPE  MYDIR:MEMOS:NOV ↵ (MYDIR:MEMOS is the path to the link file.)
```

Figure 2–4 illustrates another example.

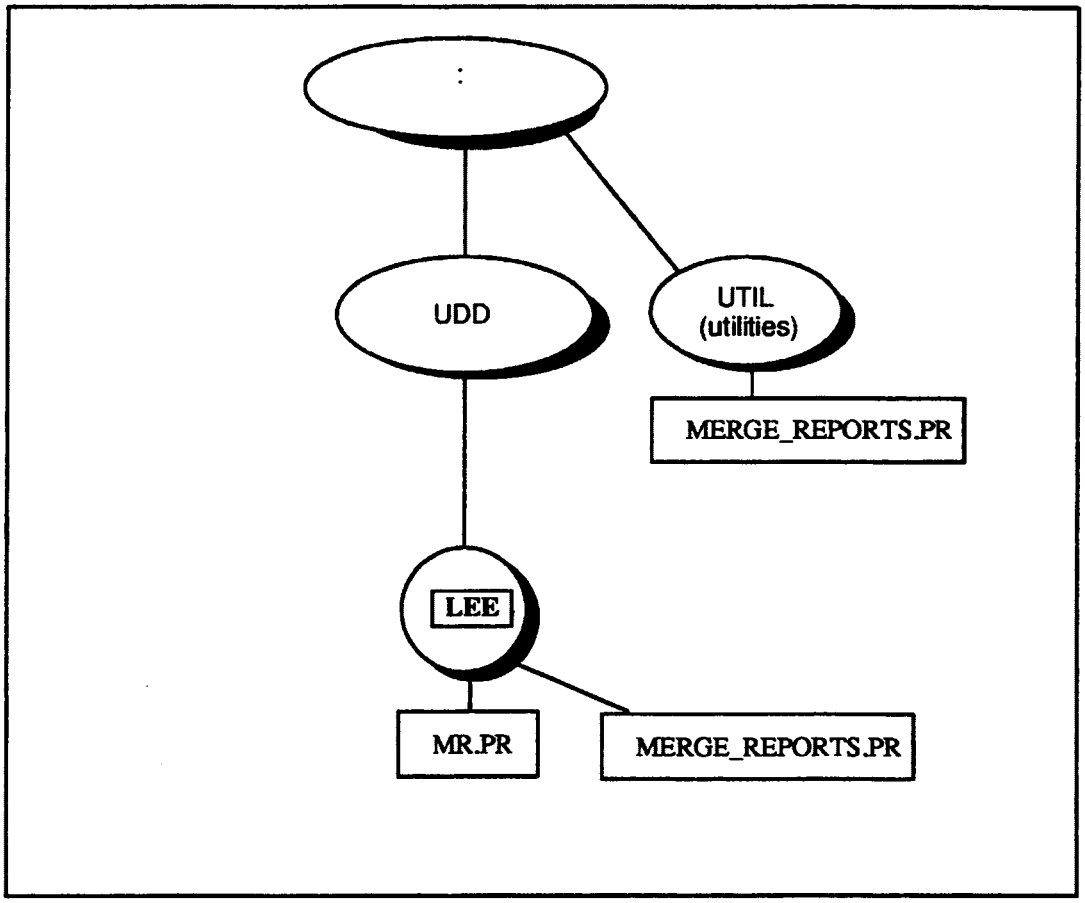


Figure 2–4 Using a Link within a Pathname

In Figure 2–4, user LEE has created a link file called MR.PR, which points to the program file MERGE_REPORTS.PR. The link file resides in LEE's initial directory (its pathname is :UDD:LEE:MR.PR).

When LEE issues the command

```
) XEQ MR ↵
```

the CLI resolves the link and performs the command

```
XEQ MERGE_REPORTS.PR
```

LEE could also use the following command (which would be necessary if :UDD:LEE is not the working directory and not on LEE's search list.)

```
) XEQ :UDD:LEE:MR ↵
```

The CLI uses the first part of this pathname (:UDD:LEE) to locate the link file.

Later, LEE wants to use another version of this program, which is in the :UTIL directory (:UTIL:MERGE_REPORTS.PR). LEE then issues the command

```
) XEQ :UTIL:MR ↵
```

and sees the message

Warning: File does not exist

An error occurred because the CLI does not simply replace the link name (MR.PR) with the associated pathname. Rather, the CLI uses the link to locate MR.PR, which does not exist in :UTIL.

If you use a pathname that includes more than one link file reference, the CLI resolves the pathname as just described, working from left to right.

Note that if a link resolves to a pathname that begins with either the prefix = or ^, the resolution of the link applies to the directory in which you use the link, *not* to the directory that contains the link file.

Again using Figure 2–4, suppose LEE had used the = prefix when specifying the link's resolution pathname with the command

```
) CREATE/LINK MR.PR =MERGE_REPORTS.PR ↵
```

Assuming LEE's initial directory (:UDD:LEE) is on LEE's search list, LEE could now execute the program in :UTIL by using the commands

```
) DIRECTORY :UTIL ↵
) XEQ MR ↵
```

The CLI locates the link file in :UDD:LEE, which resolves to =MERGE_REPORTS.PR. Since LEE's working directory is :UTIL, the prefix = applies to that directory; thus, the CLI executes the version of the merge program in :UTIL.

If LEE issues the XEQ MR command after returning to :UDD:LEE, the CLI would execute the program in LEE's directory. Similarly, the prefix ^ (uparrow) when used in a link resolution pathname, applies to the directory immediately superior to the one in which you use the link, *not* to the directory superior to the one that contains the link.

Working with the Link File Itself

Most CLI commands that open a file, when used with a link name, operate not on the link, but on the resolution file. These commands include EXECUTE, PRINT, PROCESS, TYPE, XEQ, and commands that consist of CLI macro names; they also include ACL.

Certain CLI commands, however, operate on the link file itself, *not* the file it points to. These commands include DELETE, DUMP and DUMP_II, FILESTATUS, LOA and LOAD_II, and MOVE.

For example, if you use a link file with the DELETE command, the CLI deletes the link file, not the file associated with the link. Given the files shown in Figure 2–4, if user LEE issues the command

```
) DELETE MR.PR ↵
```

the CLI deletes the link file MR.PR, not MERGE_REPORTS.PR.

Similarly, the FILESTATUS command displays the status of the link file.

```
) FILESTATUS/ASSORTMENT M+ ↵
```

Directory :UDD:LEE

MERGE_PROGRAM.PR	*TXT*	*9-Oct-90 8:43:38*	*71680*
MR.PR	*LNK*	*=MERGE_PROGRAM.PR*	

Note that if you use a pathname template that matches a link file, the CLI does not resolve the link.

Access Control

Every file in the system's directory tree has an access control list (or ACL). The ACL determines who can access the file, and also specifies the type(s) of access allowed. The system provides access control lists for security reasons. The default ACL does not allow anyone else to access your files. You can use ACLs to give other users selective access to your files; other users can do the same with their files.

There are five access types that apply to a file:

O = Owner access

W = Write access

A = Append access

R = Read access

E = Execute access

,, = Null access; none of these access types are granted.

The meaning of each access type depends on whether the file is a directory or not, as shown in Table 2–5.

Table 2–5 Access Types

Access Type	Directory File	Nondirectory File
O (Owner)	Lets user delete the directory or change its ACL.	Lets user delete the file or change its ACL.
W (Write)	Lets user insert or delete files in the directory and change their ACLs.	Lets user modify the file's contents.
A (Append)	Lets user add files to the directory.	Does not apply.
R (Read)	Lets user list the files in contents of the directory.	Lets user read the file.
E (Execute)	Lets user use directory name in a pathname; needed for access within the directory.	Lets user execute the program; applies to executable programs only.
,, (null)	Prevents user from accessing files in directory. Used with templates to prevent access by specific users.	Prevents user from accessing file. Used with templates to prevent access by specific users.

ACLs include username–access entries of the form

username,access–types (for example, CHRIS,OWARE)

(For AOS/VS II with CLI32, the username can also include a group name specification, form username:groupname,access. User groups are described later in the next major section.)

An ACL can include more than one username,access–types entry, each one specifying a username (or username template) and access types. For example,

```
) ACL MYFILE CHRIS,OWARE +,RE ↵
```

This ACL gives LEE all access and all other users Read and Execute access to MYFILE.

When you log on, the system assigns your process a *default ACL*. Every file your process creates will receive this ACL. Unless you change it, your default ACL is

your–username,OWARE (for example, CHRIS,OWARE)

This default ACL gives you complete access (all privileges) to your files, as you would expect. You can change your default ACL with the command DEFACL, explained in the next section.

You can use the ACL command to display or set the access control list for a file. For example, LEE wants to check the ACL for a newly created file.

```
) CREATE MYFILE ↵
) ACL MYFILE ↵
```

LEE,OWARE

Because the only entry in this ACL applies to LEE, LEE alone has access to the file. If LEE wants to grant Write, Read, and Execute access to SANDY, and allow all other users only Read and Execute access, LEE could issue this command:

```
) ACL MYFILE LEE,OWARE SANDY,WRE +,RE ↵
```

The system assigns access to users in the order that their usernames appear in the ACL. The access given to the first matching username will be used even if the ACL contains another matching username. For example,

JOHN+,E +SMITH,RWE JOHNSMITH,OWARE

This ACL gives all users whose username begins with Execute acces — including JOHNSMITH. Even though the third entry gives JOHNSMITH all access types, he has only Execute access to the file because his username matches a preceding entry.

Because access–types are assigned this way, be careful with ACL entries that include username templates. If there are username templates, the order of usernames should proceed from the specific to the general. Place usernames with templates *after* the ones that do not use templates. If the ACL in the example above is changed to

JOHNSMITH,OWARE JOHN+,E +SMITH,RWE

JOHNSMITH now has all access types, ADAMSMITH has Read, Write, and Execute access, and JOHNADAMS has Execute access only. Those whose usernames do not begin with JOHN or end with SMITH do not have any access to the file.

The ACL of the parent directory and the file itself determine your ability to change the file's ACL. If you have Write access to theparent directory (or O access to a file), you can change its ACL. This means that as long as the file remains in a directory to which you have Write access, you will be able to access the file.

When you assign an access control list, don't forget to give yourself access to the file. If you omit your username (either explicitly or through a template), you will not be able to access the file — a temporary nuisance.

NOTE: If you give another user Owner access to a file, that user can delete the file, regardless of the parent directory ACL.

Default Access Control List (Default ACL)

The default access control list (default ACL) is the access control list that the system automatically assigns to a file that you create. Initially, the default ACL has the form

your–username, OWARE

This gives you complete access to the file. You can change the default ACL with the DEFACL command, which specifies usernames,access–types in same form as the ACL command. For example

```
) DEFACL  CHRIS,OWARE +,E )
```

The default ACL created by this command gives CHRIS all access and all other users E access to all files this CLI process creates. Changing the default ACL is useful if you will generally want to give other users some kind of access to your files; it's easier to do this than to type ACL commands to change the ACLs of all your files. Often people change their default ACLs routinely at logon by inserting the appropriate DEFACL command in their logon macros.

The DEFACL command can display or change the default ACL. The CLI also provides the pseudomacro !DEFACL, which represents the current default ACL.

User Groups

Under AOS/VS II only, CLI32 offers user groups. A user group is a set of users, associated by group name. Groups offer a simple way to handle access control for projects, since people can gain access to files by groupname, not username. Groups streamline file access control for group–oriented tasks like projects. Their main benefit is that they can eliminate the maintenance of long, intricate ACLs.

For your system to use groups, someone acting as system manager must create a directory with pathname :GROUPS (needed only once). Then within this directory someone must create a file for each group that contains the usernames of people allowed to join the group. Creating the :GROUPS directory and group files is detailed in *Managing AOS/VS and AOS/VS II.*

This section explains how users can use groups, not how to create the files. It explains

- Setting up ACLs of files to allow group access;
- Joining a group using the GROUPLIST command.

Details on these tasks follow.

Setting up ACLs to Allow Group Access

Regardless of your membership in a group, your default ACL remains as your username,OWARE, or the value set with the DEFACL command. Any files you create in the group directory structure will receive your default ACL. Depending on this default ACL, other group members may not be able to access such files. (Normally you will not want to change your default ACL at logon to grant access to group members, since this would allow group members to access all files you create, even those in your user directories.)

For groups to work properly, the ACLs of files the group will use must allow access to group members. This does not occur by default — usually, users who are group members will need to make it happen, perhaps by changing their default ACLs when they join the group (so files they create will be accessible to group members).

The form to specify group access in an ACL is

username:groupname, access *[...]*

Template characters are allowed in both username and group name. For example, assume that there is a group named MARK_II. The ACL +:MARK_II,OWARE allows access to all members of the group. As usual with ACLs, the username lets you allow different kinds of access to different members of the group. For example, the ACL

JKM:MARK_II,WARE +:MARK_II,RE

gives user JKM WARE access to the file(s) and other members of the group RE access.

The order of username:group specifications in ACLs works the same way as the order of usernames. If you use template characters, you must order the username and group name specifications carefully to get the results you want. Generally, you will want to order them from specific to general. For example, take the following ACL:

```
JKM,OWARE  CSTONE:MARK_II,WARE  +:MARK_II,RE  +,E
```

This ACL gives user JKM all access to the file; the OWARE specification, since it comes first, overrides any other specification (group or nongroup) for JKM. The ACL gives CSTONE WARE access the file when she is a member of group MARK_II; it gives other members of group MARK_II RE access; and it gives all other users E access.

Joining a Group Using the GROUPLIST Command

All groups that a user has joined are kept in a group list, which is analagous to a search list. At logon, your group list is empty; it contains no group names. You can learn the names of available groups from the system manager. Or if you have read access to :GROUPS, you can learn the names of all groups by typing

```
) FILESTATUS  :GROUPS:+ )
```

To join a group, use the GROUPLIST command with the group name as an argument. GROUPLIST can also display the group names that you have joined. For example

```
) GROUPLIST  MARK_II )
) GROUPLIST )
```

MARK_II

A group list can include up to eight group names. GROUPLIST command switches let you insert or remove group names in your group list, or delete your group list.

If you try to join a group that does not include your username, the system will display the error message *User cannot be in group*. If the group does not exist, it will display *Group does not exist*. (If the :GROUPS directory doesn't exist, the system will display *GROUPS directory does not exist;* in this case, you may want to contact the system manager.)

The group list, like the search list, is part of the CLI environment; you can specify a different group list on each level. A pseudomacro, !GROUPLIST, displays the current group list.

Group Access Example

This example assumes that your username is CSTONE, that you are a member of group MARK_II, and that users with username JKM and CSTONE are also members of group MARK_II.

To begin, you log on.

```
) DIR :UDD:MARK_II ↵          (You make MARK_II the working directory.)

) FILES/AS/S ↵                (Try to list files.)

Error: Read access is required   (Receive access error message.)

) GROUPLIST MARK_II ↵         (Join group MARK_II.)

) FILES/AS/S ↵                (Again try to list files.)

PHASE.STAFFING                (You have joined the group, so system obeys
...                            command and lists files.)

) SED PHASE2.OVERVIEW ↵       (Run the SED editor to create and edit a file.)
...                           (Edit the file and exit from SED.)

) ACL/V PHASE2.OVERVIEW ↵                (Check the ACL of the file.)

PHASE2.OVERVIEW CSTONE,OWARE             (Your default ACL does not allow
                                          access to other group members.)

) ACL/V PHASE2.OVERVIEW [!USERNAME],OWARE & ↵
&) JKM:MARK_II,WARE +:MARK_II,RE +,E ↵   (Add access for group members:
                                          WARE for JKM and RE for other
                                          members.)

) DEFACL PHASE2.OVERVIEW [!USERNAME],OWARE & ↵
&) JKM:MARK_II,WARE +:MARK_II,RE +,E ↵   (Change your default ACL to allow
                                          group access to other files you create
                                          during this session. You could put
                                          this DEFACL command after the
                                          GROUPLIST command in a macro;
                                          you would execute the macro when
                                          you wanted to join the group.)

...                                      (You continue to work in the group
                                          directory structure.)

) DIR/I MEMOS ↵                          (Return to your user directory
                                          structure.)

) GROUPLIST/KILL ↵                       (Set your group list to null,
                                          preventing access to group files to
                                          which your username alone does not
                                          grant access.)
```

```
) DEFACL [!USERNAME],OWARE +,E ↵
```
(Restore your original default ACL. You could put the GROUPLIST/K command and this one in a macro to execute when you want to leave the group.)

...

(You continue to work in your user directory structure.)

File Permanence

To help you protect your files, the CLI lets you define any file as a *permanent* file. The system will not delete a permanent file except under certain conditions. This safeguard helps prevent the accidental deletion of important files.

To turn permanence on for a file, use the PERMANENCE command, and specify the file (or a template) and the ON option. For example

```
) PERMANENCE MYFILE ON ↵
```

If you try to delete a permanent file, the CLI returns an error and retains the file.

```
) DELETE/V MYFILE ↵
```

Warning: Cannot delete permanent file, File MYFILE

Permanence is helpful especially when you use a filename template with the DELETE command. For example, you might use the following command to delete the files MYPROG.PR, MYPROG.ST, and MYLISTING.

```
) DELETE MY+ ↵
```

This template also matches MYFILE, which you do not intend to delete. If you made MYFILE permanent, however, the CLI does not delete it.

If you later decide you want to delete a permanent file, you can turn its permanence off. For example

```
) PERMANENCE MYFILE OFF ↵
) DELETE/V MYFILE ↵
```

Deleted MYFILE

NOTE: With AOS/VS, if a directory contains one or more permanent files, and you delete that directory, the permanent files are lost. AOS/VS II will not let delete a directory that contains permanent files; instead it will return the error message *Directory delete error.*

Generic Files

The system has several *generic* files, which simplify the use of common files and devices. A generic file is essentially a name that the system associates with an actual file or device. This assignment can change from time to time, allowing the use of a single name to refer to different files.

The purpose of a generic file is to allow you (or a program) to use a standard name, while the system interprets this name as meaning an actual file. Thus a program can be designed and coded to write to a generic file, and you can assign the generic file to a different disk file each time you run the program. The program then outputs data to different files, without your having to rewrite the program each time. The program can run without being tied to a particular file or device.

All generic files begin with the pathname prefix @. This prefix represents the :PER directory, which contains the generic files. Table 2–6 describes the generic files.

Table 2–6 Generic Files

Filename	Description
@CONSOLE	Represents your terminal (console).
@INPUT	When you are working interactively, @INPUT corresponds to your terminal's keyboard. In batch mode, this generic file corresponds to a disk file.
@OUTPUT	When you are working interactively, @OUTPUT corresponds to your terminal's display unit. In batch mode, this generic file corresponds to a disk file.
@LIST	Represents a listing file, which you normally use for recording program messages or other output. You can assign a disk file as your list file, and then use the /L switch with CLI commands to direct the output to the list file. Use the LISTFILE command to display or set your current list file. In batch mode, the system creates a disk file for @LIST. By default, when the batch job is done, the system then sends this disk file to the LPT queue and deletes the file.
@DATA	Represents a data file, which you normally use for input. You can assign a disk file as your current data file and then run a program that is designed to read the generic @DATA file. Use the DATAFILE command to display or set your current data file.
@NULL	Unlike the other generic filenames, @NULL does not correspond to an actual file. It is useful within programs to test I/O statements. If you send data to @NULL, the system discards the data; if you try to read @NULL, an end–of–file condition occurs.

Device Names and Queue Names

Like generic filenames, device names and queue names reside in the :PER directory. All device and queue names, therefore, begin with the pathname prefix @.

In AOS/VS, device names are rigidly defined; they cannot be changed. In AOS/VS II, the system manager can assign any legal filename to any device during system generation. However, AOS/VS II does have default device names, and these are identical to their AOS/VS counterparts.

Table 2–7 lists AOS/VS device names, AOS/VS II default device names, and queue names.

Table 2–7 Device and Queue Names

Device/Queue Name	What It Means
@CONn	Console n device name (@CON1, @CON2, and so on). A console can be a terminal, a modem line, or printer attached to an asynchronous controller.
@DKB or @DKB1	First or second fixed–head disk controller device name.
@DPxn or @DPx1n	Disk unit device name on first or second moving–head disk controller (x indicates type and n indicates unit number).
@LFD	Labeled diskette device name (AOS/VS only).
@LMT	Labeled magnetic tape device name.
LPT or LPT1	First or second line printer queue name.
LQP or LQP1	First or second letter–quality printer queue name.
@MRCTAPEccssuu	Tape unit device name on Message–based Reliable Channel (MRC). The cc indicates the MRC chassis number, ss the slot number occupied by the controller, and uu the unit number — all in hexadecimal. The default name of the first MRC tape unit is MRCTAPE000A00.
@MRCDISKccssuu	Disk unit device name on Message–based Reliable Channel. See comment under MRCTAPE. The default name of the first MRC disk unit is MRCDISK000E00.
@MTB0n or @MTB1n	Tape unit device name on first or second MTB controller; n indicates the unit number on the controller.
@MTCn or @MTC1n	Tape unit device name on first or second MTC controller.
@MTDn or @MTD1n	Tape unit device name on first or second MT controller.
@MTJn or @MTJn	Tape unit device name on first or second MTJ controller.
PLT or @PLT1	First or second plotter queue name (AOS/VS only).
@VCONn	Network virtual console device name.

File Types

The operating system uses many different kinds of files, such as text files, program files, log files, device files, and so on. In fact, the operating system predefines 128 unique file types (numbered 0 through 127), and allows you to define 128 more (numbers 128 through 255).

Table 2–8 lists the file types by number, and gives a brief description of what each type means.

Table 2–8 File Types

Number	Type Code	Type
0	LNK	Link file
1	SDF	System data file
2	MTF	Magnetic tape file
3	GFN	Generic filename
4–9	—	(Reserved)
10	DIR	Standard disk directory
11	LDU	Logical disk unit (LDU)
12	CPD	Control point disk directory
13	MTV	Magnetic tape volume
14	MDR	Reserved for AOS/RT32 memory directory
15–19	—	(Reserved)
20	DKU	Disk unit
21	MCU	Multiprocessor communications unit
22	MTU	Magnetic tape unit
23	LPU	Data channel line printer
24	LPD	Data channel line printer 2
25	LPE	Data channel laser printer
26–29	—	(Reserved)
30	IPC	IPC port entry
31	—	(Reserved)
32	SPR	Spoolable peripheral directory
33	QUE	EXEC queue entry
34	LMT	Labeled magnetic tape
35	GLT	Labeled media
36	TRA	Paper tape reader
37	CRA	Card reader
38	—	(Reserved)
39	—	(Reserved)
40–41	—	(Reserved)
42	TPA	Paper tape punch
43	PLA	Digital plotter
44	LPA	Programmed I/O line printer
45	LP2	Programmed I/O line printer 2

(Continued)

Table 2–8 File Types

Number	Type Code	Type
46–48	—	(Reserved)
49	CON	Console (hard copy, CRT, virtual console, or PC console)
50	—	(Reserved)
51	RMA	Remote host — RMA access
52	HST	Remote host — X.25 SVC access
53	NPN	Network process name
54	PVC	Remote host — X.25 PVC access
55–59	—	(Reserved)
60	SYN	Synchronous communication line
61	—	(Reserved)
62	LUG	System Network Architecture Logical Unit Group
63	—	(Reserved)
64	UDF	User data file
65	PRG	Program file (AOS)
66	UPF	User profile file
67	STF	Symbol table file
68	TXT	Text file (default type created by SED and SPEED text editors and the type to choose when exporting from CEO)
69	LOG	System log file (accounting file)
70	NCC	FORTRAN carriage control file
71	LCC	FORTRAN carriage control file
72	FCC	FORTRAN carriage control file
73	OCC	FORTRAN carriage control file
74	PRV	AOS/VS or AOS/VS II program file
75	WRD	Word processing file
76	AFI	APL file
77	AWS	APL workspace file
78	BCI	BASIC core image
79	DCF	Device configuration file (networking)
80	LCF	Link configuration file (networking)
81	LUG	SNA logical unit group
82–86	—	(Reserved)
87	UNX	UNIX file (default type of file created on a UNIX system)
88	BBS	Business BASIC save file
89	VLF	Business BASIC volume label file

(Continued)

Table 2–8 File Types

Number	Type Code	Type
90	DBF	Business BASIC database file
91	GKM	DG graphics kernel metafile
92	VDM	Virtual device metafile
93	NAP	NAPLPS standard graph file
94	TRV	TRENDVIEW command file
95	SPD	Spreadsheet file
96	QRY	PRESENT query macro
97	DTB	CEO data table
98	FMT	CEO format file
99	WPT	CEO text interchange format
100	DIF	CEO data interchange format
101	VIF	CEO voice image file
102	IMG	CEO facsimile image file
103	PRF	CEO print ready file
104	PIP	Pipe file
105	TTX	Teletex file
106–127	—	(Reserved)
128–255	—	User–defined file types

(Concluded)

With certain CLI commands, you can specify a type of file, usually by appending the /TYPE= switch. Depending on the command, you can supply this switch with either a file type number or code. See Table 2–8 for a list of file type numbers and codes.

For example, to create a file with the user–defined type 205, you could use the command

```
) CREATE/TYPE=205 MYFILE ↵
```

You can use the /TYPE= switch to list, dump, load, or move files of a specific type. For example, to list all the files in your working directory that have the user–defined file type 205, you could type

```
) FILESTATUS/TYPE=205 ↵
```

Or, to list all the directory files in your working directory, you could type

```
) FILESTATUS/ASSORTMENT/TYPE=DIR/TYPE=CPD ↵
```

(The /ASSORTMENT switch causes the CLI to display the file type with each filename it lists.)

You can use the /TYPE= switch during dump or load operations to restrict the operation to specific file types.

End of Chapter

Chapter 3
CLI Environment Settings

This chapter explains the settings that govern the CLI environment in which you work. Information about how to use these environment levels appears in the following major sections:

- About Environment Levels
- Working Directory
- Search List
- Default Access Control List
- Group List
- CLI Prompt
- Super Privilege Modes
- List File
- Data File
- User Log File
- CLI Variables
- CLI String(s)
- Device Characteristics
- Prefix
- Screenedit Mode
- Squeeze Mode
- Trace Mode
- Exception Conditions
- Examples of Using CLI Environment Levels

About Environment Levels

When you use the CLI, many factors define your working context. These factors (such as your working directory, search list, and default access control list) constitute your CLI *environment.* Your current environment settings affect how certain CLI commands work and define the scope of your activity.

To provide flexibility in your working environment, the CLI allows multiple CLI environment *levels.* You can use different environment parameter settings at each level without affecting the settings of other levels.

Multiple levels are especially useful when you want to make temporary changes to your CLI environment; for example, change your wokrting directory or search list. You can later restore the previous settings simply by returning to the previous environment, where the settings haven't changed. Many macros, in fact, are designed to execute in a different level so that they will not affect your current environment settings.

Table 3–1 lists the settings that make up each level of your working environment, and shows the CLI command(s) that you can use to display or change the setting. The rest of this chapter describes each of the CLI environment settings.

Table 3–1 CLI Environment Settings

Environment Setting	CLI Command(s)
Class 1 exception	CLASS1
Class 2 exception	CLASS2
CLI prompt commands	PROMPT
CLI String	STRING
Device characteristics	CHARACTERISTICS
Data file	DATAFILE
Default ACL	DEFACL
Environment level	LEVEL to display; PUSH, POP, or /PREVIOUS switch to change
Group list	GROUPLIST (AOSVS II and CLI32 only)
List file	LISTFILE
Log file	LOGFILE
Prefix (CLI32 only)	PREFIX
Screenedit mode	SCREENEDIT
Search list	SEARCHLIST
Superprocess mode	SUPERPROCESS
Superuser mode	SUPERUSER
System Manager mode	PRIVILEGE/SYSTEMMANAGER (CLI32 only)
Squeeze mode	SQUEEZE
Trace mode	TRACE
Variables	VARn VAR (CLI32 only)
Working directory	DIRECTORY

When you first log on, the system uses its default values for the CLI environment settings. If your user profile specifies a macro file to be executed when you log on (an initial message file), you can include the appropriate CLI commands in the macro to set up your environment the way you want it.

For detailed information about these CLI commands, refer to the command descriptions in Chapter 5.

Level Numbers

When you first enter the CLI, you are at Level 0, the highest environment level. As you move down to a new level, the level number increases; as you return to a previous level, the level number decreases.

To display the current environment level, use the CLI command LEVEL; to change levels, use either the PUSH or POP command.

- PUSH moves you down to the next level (to Level 1 from Level 0, or to Level 2 from Level 1, for example). The new level automatically takes on the environment settings of the previous level (the one you just left).
- POP moves you up to the previous level (to Level 1 from Level 2, or to Level 0 from Level 1, for example), restoring the environment settings for that level. Note that any setting changes you made in the higher level are now lost.
- The /LEVEL or /PREVIOUS switch, available on certain commands like DIRECTORY, specifies the value that command had on a different level. Either switch lets you specify a the value of a previous level without the POP command (POP eliminates values on the level you pop from).

To display the environment settings for the current level, use the CURRENT command. The CLI also provides the pseudomacro !LEVEL, which represents the current level number. The PREVIOUS command displays the settings for the next higher level (that is, the level you would POP to). Because Level 0 is the highest environment level, you cannot use the POP or PREVIOUS commands, or the /PREVIOUS switch, in Level 0.

Working Directory

The working directory is your current position within the directory tree. (For details on the filw system and directory structure, see Chapter 2.) The working directory determines which files are immediately available to you. If you refer to a file without giving a full pathname, the CLI assumes that you are referring to a file in the working directory.

You can use the DIRECTORY command to display the name of the working directory, or move to a different directory. The CLI also provides the pseudomacro !DIRECTORY, which represents the full pathname of your working directory.

Search List

Unless you give a full pathname when referring to a file, the system assumes you are referring to a file in your working directory. Your search list is a list of directories that the system searches when it can't find a file in the working directory. (Chapter 2 provides more information about your search list.)

You can use the SEARCHLIST command to either display the current search list, or to change it. The CLI also provides the pseudomacro !SEARCHLIST, which represents the current search list.

Default Access Control List (Default ACL)

The default access control list (default ACL) is the access control list that the system automatically assigns to any file that you create. Initially, the default ACL has the form

your–username, OWARE

This gives you complete access — and precludes access by other users — to files you create. You can change your default ACL if you want to permit or restrict file access to other usersindividuals. (See Chapter 2 if you need more information about ACLs and the access types.)

You can use the DEFACL command either to display the current default ACL, or to change it. The CLI also provides the pseudomacro !DEFACL, which represents the current default ACL.

Group List

The group list is a list of user groups that you have joined via the GROUPLIST command. For example,

```
) GROUPLIST FUTURES ↵
```

When you log on, your group list is empty. Joining a group gives you access to files that you might not otherwise be able to access. You can join a group only if your username is included in a file named for the group in the groups directory. The CLI also provides the pseudomacro !GROUPLIST, which represents the current group list.

Groups are further explained in Chapter 2 and the GROUPLIST command in Chapter 5; creating groups is described in *Managing AOS/VS and AOS/VS II.* (Groups are available under AOS/VS II with CLI32 only.)

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.

Prefix

The prefix is the character(s) displayed as the CLI prompt. (The CLI prefix has no relation to pathname prefix characters like the caret, ^.) The initial prefix is a right parentheses,). You may never want to change this. But if you want to, you can change the prefix via the PREFIX command to a string of up to 24 characters.

One reason to change the prefix involves adding information. If your system is part of a network and you would like the CLI prompt on each host to identify the host, you can change the prefix to include the hostname. For example, on a remote host named CYRANO, your log–on macro might include the command

```
PREFIX CYRANO[!ASCII 251]
```

(The pseudomacro [!ASCII 251] expands to a right parenthesis.) When you log on to CYRANO, your CLI prompt is

CYRANO)

With CLI32, along with the prefix, you can specify a different setting for the number of command lines saved in the HISTORY buffer. In CLI32, both the prefix and the number of lines in HISTORY are retained as part of the environment level. (The prefix is part of the environment in CLI32 only. In CLI16, the prefix is not part of the environment level; it remains constant through all levels.)

Screenedit Mode

Screenedit mode applies to video display terminals only. When Screenedit mode is turned on, you can use control characters and cursor control keys to move the cursor and to edit the current input line. (For a list of Screenedit control characters, see Chapter 1.)

When Screenedit mode is turned off, most cursor control keys and Screenedit control characters no longer affect the cursor. Instead they appear on your terminal as ^A, ^B, and so on. The Screenedit control characters CTRL–I and CTRL–L continue to work as do their keyboard counterparts, TAB and ERASE PAGE.

You can use the SCREENEDIT command either to display the current Screenedit mode or to turn it on or off.

Squeeze Mode

Squeeze mode, when turned on, eliminates excess blanks and tabs from information that the CLI displays. In other words, it squeezes output so that a single blank appears between each item in the output line.

When Squeeze mode is off, as it normally is, the CLI arranges the output for certain commands so that information is aligned in columns for easy reading.

You can use the SQUEEZE command to display the current Squeeze mode setting, or to turn it on or off. All CLI commands accept the /Q switch, which you can use to turn on Squeeze mode for the duration of that one command.

This switch affects only text that the CLI itself displays. For instance, text that the FILESTATUS command displays will be squeezed, but not text printed as a result of the QPRINT command.

NOTE: Squeeze mode never affects the format of text that the TYPE command displays. TYPE output is never squeezed.

You may find the /Q command switch useful when you are building a file of filenames and want only one space between names. For example, if you want to assemble many files, you could begin with the commands

```
) DELETE/2=IGNORE SOURCE_NAMES ↵
) FILESTATUS/NHEADER/SORT/Q/L=SOURCE_NAMES  +.SR ↵
```

These commands have the CLI delete any file named SOURCE_NAMES. Then they tell it to create a file named SOURCE_NAMES and place in it the names of all files whose names end with .SR. Furthermore, the filenames are sorted in alphabetical order and they are squeezed together so that only one space is between each filename.

Next, you edit the freshly created file SOURCE_NAMES and add an ampersand line–continuation character (&) to the end of each line. Its contents would look something like the following.

```
=HER_PROGA.SR =HIS_PROGA.SR =MY_PROG1.SR =MY_SUB1.SR&
=MY_SUB2.SR =YOUR_PROG1.SR&
```

The ampersand at the end of the last line is optional. You could then use the file of filenames as follows:

```
) XEQ MASM [SOURCE_NAMES] ↵
```

This command assembles all the files whose names appear in file SOURCE_NAMES.

Trace Mode

Trace mode, when turned on, lets you track the execution of CLI commands, macros, or pseudomacros. The CLI reports each traced item, showing the expansion of any arguments, before executing it. You can use tracing to examine the execution of the statements within a macro.

You can use the TRACE command to display the current Trace mode, or to turn tracing on or off for commands, macros, or pseudomacros.

Exception Conditions

If you enter an invalid command line or if you try to execute a syntactically correct line in an improper environment, you create an *exception condition* (error). Exception conditions fall into one of two classes, Class 1 or Class 2, described next.

The CLI response to an exception depends on the setting that you've defined for that class of exception. Table 3–3 lists and describes the settings that you can use.

Table 3–3 Exception Condition Settings

Setting	Effect
IGNORE	None. The CLI ignores the exception condition and continues processing as best it can.
WARNING	The CLI displays a warning message and continues processing. The only difference between IGNORE and WARNING is the message.
ERROR	Execution ceases for the current command and the CLI displays an error message. In macros and multiple–command input lines, the CLI discards any remaining commands.
ABORT	ABORT terminates the CLI and control returns to the CLI's father. If the CLI's father is EXEC, the system logs you off.

Class 1 Exceptions

When a command that would change any CLI environment parameter fails, the CLI returns a Class 1 exception condition. For example, if you issue the POP command while in Level 0, a Class 1 exception condition occurs. Likewise, if you try to change your working directory to a nonexistent directory or to a directory to which you do not have Execute access, a Class 1 exception condition occurs.

The action that the CLI takes on Class 1 exception conditions depends on the current Class 1 setting. You can set Class 1 to any of the values shown in Table 3–2. By default, the Class 1 setting is ERROR in interactive mode, and ABORT in batch jobs.

The CLASS1 command lets you display the current Class 1 setting or change it. If you want to override the current Class 1 setting for the duration of a single CLI command, use the /1= switch with the command.

Class 2 Exceptions

When a command that would not alter the CLI environment fails, the CLI returns a Class 2 exception condition. When such a command fails, the CLI responds with whatever action the Class 2 parameter is set to. For example, if you try to rename a nonexistent file, a Class 2 exception condition occurs. Or if you issue a DELETE command for a file without Write access to its parent directory, a Class 2 exception condition occurs.

The action that the CLI takes on Class 2 exception conditions depends on the current Class 2 setting. You can set Class 2 to any of the values shown in Table 3–3. By default, the Class 2 setting is WARNING (for both interactive and batch modes).

The CLASS2 command lets you display the current Class 2 setting or change it. If you want to override the current Class 2 setting for the duration of a single CLI command, use the /2= switch with the command.

NOTE: For the following four exception conditions — CLASS1=IGNORE, CLASS1=WARNING, CLASS1=ERROR, and CLASS2=ERROR — the CLI will compare your starting level with your current level when you encounter an error. If they are different, the current level will be output. You cannot suppress this output.

Examples of Using CLI Environment Levels

The following two examples suggest ways that you can use CLI environment levels.

Example 1

The macro SAMPLE.CLI illustrates the use of several commands described in this chapter. After moving to a new environment level (via PUSH), the macro changes several environment settings. Before completing its execution, the macro uses the POP command to return to the original environment level, thereby restoring all the original settings. Figure 3–1 shows the contents of the SAMPLE.CLI macro, and numbers each line for use in explanation.

```
1   PUSH; PROMPT POP
2   SQUEEZE OFF
3   CLASS1 ABORT
4   CLASS2 WARNING
5   LISTFILE %1%
6   DATAFILE %2%
7   DIRECTORY :UDD:USER:MDIR
8   SEARCHLIST :UDD:USER:NDIR [!SEARCHLIST]
9   XEQ/S MYPROG1
10  XEQ [!STRING]
11  POP
```

Figure 3–1 Macro SAMPLE.CLI

Note that this macro begins with the PUSH command and ends with the POP command. This sequence of commands allows you to change the CLI environment for the duration of the macro, without affecting the original environment. (The PROMPT POP command in Line 1 ensures that you will return to the original environment even if the macro execution is interrupted.)

Line 2 ensures that Squeeze mode is off; that is, the CLI does not compress output. Lines 3 and 4 set the Class 1 and Class 2 exception conditions to appropriate severity levels for the duration of the macro.

The arguments to the LISTFILE command on line 5 and the DATAFILE command on line 6 are called *dummy arguments*. The CLI replaces these with actual argument that you supply when you call the macro. For example, if you call the macro as follows

```
) SAMPLE @LPT DATA1 ↵
```

The line printer (@LPT) becomes the list file, and the disk file DATA1 becomes the data file. Here, the CLI replaces the dummy arguments %1% and %2% in the macro with the actual arguments @LPT and DATA1 respectively. Lines 5 and 6 effectively become LISTFILE @LPT and DATAFILE DATA1 respectively.

Line 7 sets the working directory to MDIR. Line 8 sets the search list to include the pathname to NDIR, followed by the contents of the current search list. Line 9 invokes the user program MYPROG1. The /S switch directs the termination message for MYPROG1 to the CLI String.

Suppose you've designed MYPROG1 to return either the string MYPROG2 or MYPROG3 upon completion. This string indicates the next program to be executed. So, after MYPROG1 terminates, the CLI executes either MYPROG2 or MYPROG3, depending on the value of the CLI String. (See line 10.)

After either MYPROG2 or MYPROG3 executes, the macro issues the POP command to return to the previous environment and restore the environment settings that existed when you called the macro.

Example 2

In this example, Chris is working in the directory :UDD:CHRIS:TEST. But Chris wants to check on a file in another directory, :UDD:COMMON:PROGRAMS. Rather than using the DIRECTORY command to change the working directory to :UDD:COMMON:PROGRAMS, and then using it once again to return to :UDD:CHRIS:TEST, Chris decides to use CLI environment levels.

First, Chris displays the working directory and CLI environment level.

```
) DIRECTORY ↵
:UDD:CHRIS:TEST

) LEVEL ↵
Level 0
```

Next, Chris moves down the next environment level, and sets the working directory.

```
) PUSH ↵
) LEVEL ↵
Level 1

) DIRECTORY :UDD:COMMON:PROGRAMS ↵
) DIRECTORY ↵
:UDD:COMMON:PROGRAMS
```

Chris now uses the FILESTATUS command to obtain a listing of all files that begin with the letters RE.

```
) FILESTATUS/SORT/ASSORTMENT RE+ ↵
Directory :UDD:COMMON:PROGRAMS
REGION.RPTS        TXT      17-Apr-90  9:46:08    13825
RELEASE_1.5        UDF      18-Nov-89  9:57:40    36778
REVIEW.PR          PRV      17-Apr-90 15:17:22    49152
```

Finally, Chris uses the POP command to return to the previous CLI environment. All settings for that environment are restored.

```
) POP ↵
) LEVEL ↵
Level 0

) DIRECTORY ↵
:UDD:CHRIS:TEST
```

End of Chapter

Chapter 4
CLI Macros

This chapter explains CLI macros and how to create and use them. Major sections proceed as follows.

- About Macros
- Using Dummy Arguments
- Specifying Switches
- Using Parentheses and Brackets in Macros
- Using Conditional Pseudomacros
- Macro Examples: The Calculator and Space Percentage Macros

About Macros

A *macro* is a file that contains one or more CLI commands. By typing the name of the macro file, you can execute the commands in that file. Macros, therefore, let you execute a sequence of commands by typing a single command.

Macros provide a convenient alternative to individually executing a series of related commands. You can use macros to streamline your day–to–day operations, and to build special tools for your own use. Instead of giving inexperienced users a series of CLI commands to type, you can place the commands in a file and then the users only have to type the name of the file.

Suppose, for example, that you have a file called TDP.CLI, which contains the following CLI commands to display the time, date, and your process ID:

```
COMMENT This is macro TDP.CLI.
TIME
DATE
WRITE Your process number is [!pid].
```

You can execute the commands in this macro by simply typing the filename, with or without the .CLI suffix:

```
) TDP ↵
12:05:33
26–Jan–91
Your process number is 15.
```

This process of executing the commands in a macro file by specifying the filename is known as *calling*, *invoking*, or *running* a macro.

Naming a Macro

Because a macro is a file, any legal filename is valid as a macro name. By convention, however, the CLI expects a macro to have the filename suffix .CLI. Although you do not need to use .CLI in filenames of macros you create, you will find this suffix useful for distinguishing macros from other files.

For example, you can use the command

```
FILESTATUS +.CLI
```

to display the status of all the files in your working directory that have the .CLI suffix. This is a useful method for locating your macro files.

To execute the macro, you can omit the .CLI suffix. Because the CLI expects a macro filename to end in .CLI, it first looks for a file with the name you typed *and* the .CLI suffix. If the CLI cannot find that file, it searches for the filename you typed. When the CLI finds either file, it tries to execute the commands in the file.

So, if you type

```
) TDP ↵
```

the CLI first looks for file TDP.CLI; if that file does not exist (either in your working directory or a directory on your search list), the CLI looks for a file named TDP.

Macro Names and CLI Commands

If a macro name (without the .CLI suffix) is the same as a CLI command or a valid abbreviation of one, the CLI executes the command and not the macro. As a rule, you should not assign macro names that will conflict with CLI command names. If such a conflict arises, however, you can do one of the following:

1. Include the .CLI suffix when calling the macro.

   For example, if you have a macro called TIME.CLI, you can ensure that the CLI will execute your macro rather than its TIME command if you type

   ```
   ) TIME.CLI ↵
   ```

2. Specify the complete pathname for the macro (with or without the .CLI suffix). You can do this with a pathname prefix (including the characters ^, :, or =).

   For example, if the macro TIME.CLI is in the working directory, you can execute it if you type

   ```
   ) =TIME ↵          (or =TIME.CLI)
   ```

3. Enclose the macro name with square brackets (to force the CLI to execute the contents of the macro file). Square brackets tell the CLI to use the *contents* of the enclosed value rather than the value itself. So you could also execute a macro called TIME.CLI by typing

   ```
   ) [TIME] ↵  (or [TIME.CLI])
   ```

Creating a Macro

To create a macro, enter into a file the commands you want to execute in the order you want to execute them. Place each command on a separate line (or use a semicolon to separate multiple commands on a single line). Give the macro file a name that will be convenient to use (brief), and that will suggest the purpose of the macro.

Use the COMMENT statement to place comments in your macro. These comments can identify the macro, make it easier for yourself to understand it, and make it easier for someone else to understand it (since other people may want to adapt it for their own needs). Unfortunately, you cannot use parentheses, brackets, or semicolons in COMMENT statements because the CLI will try to interpret them. With CLI32, you can use lexical comments (\\), which the CLI does not interpret. Using comments is further explained in Chapter 1.

The usual way to create a macro file is to use a text editor (such as SED or SPEED). If you are in the process of developing a macro, or changing an existing macro, you should use a text editor. This method gives you the opportunity to edit command lines and make changes as necessary.

If you are certain about the contents of a macro, and the macro is relatively short, you can use the CLI CREATE command with the /I switch to create the macro. This method lets you create the macro file, and enter commands directly into it; this method, however, restricts editing to the current input line.

When you type CREATE/I and supply the name of the macro file you want to create, the CLI displays the prompt

))

This prompt tells you that the CLI is waiting for input. Each line you type (and end by pressing NEW LINE) is placed in the newly created macro file. (The CLI does not execute the command or check its syntax.) After you have entered the last line of the macro, type a single right parenthesis, and then press the NEW LINE key.

For example, to create the TDP.CLI macro previously listed, you would type

```
) CREATE/I TDP.CLI ↵
)) COMMENT This is macro TDP.CLI. ↵
)) TIME ↵
)) DATE ↵
)) WRITE Your process number is [!pid]. ↵
))) ↵
)
```

If you rename a macro that mentions its own filename literally (as this one does), be sure to update all internal occurrences of the old filename.

Formatting a Macro

Generally, the statements that make up a macro follow the CLI syntax rules described in Chapter 1. The following three major sections describe special syntax and usage that apply to macros. These sections describe

- Using dummy arguments to represent actual arguments.
- Using parentheses to expand a series of arguments.
- Using square brackets to specify the contents of an item rather than the item itself.

Using Dummy Arguments

The commands within a macro file can contain specific arguments. A certain macro, for example, might include the command DELETE FILEA. When you execute it, the macro deletes FILEA, but no other file. To delete another file, you would have to edit the macro.

Often you will want your macro to be more flexible, so that you can use it in different situations with different arguments. To enable a macro to accept arguments at the time a CLI command calls the macro, you can use *dummy arguments* within the macro. A dummy argument marks a place in a command line in a macro where the CLI will substitute an actual value that you supply when you call the macro. (Dummy arguments apply to macros only; you cannot use them when entering a CLI command directly.) The general format for a dummy argument is a number surrounded by percent signs. The number represents the numeric position of the argument on the command line used to invoke the macro. The following macro statement, for example, deletes the file you specify as the macro's first argument:

```
DELETE %1%
```

So, if the macro is named DELETE_1.CLI and you give the command

```
) DELETE_1 FILEA ↵
```

the macro statement

```
DELETE %1%
```

becomes

```
DELETE FILEA.
```

The symbol you use between the percent signs can represent

- A single argument.
- A range of arguments.
- One or more switches with or without switch arguments.

The next few sections explain the format and use of single dummy arguments, range dummy arguments, and switch dummy arguments.

Specifying a Single Argument

The simplest type of dummy argument contains a single number enclosed by percent signs. Its format is

%n%

The number n represents the numeric position of the actual argument when you call the macro. For instance, %1% represents the first argument, %2% represents the second argument, %3% represents the third argument, and so on.

The dummy argument %0% stands for the macro name (which is useful if you want the macro to call itself). Using %0% instead of the actual macro name enables the macro to work even if it is later renamed.

Dummy Argument Example

The macro SWITCH.CLI is designed to exchange the names of two files. It contains these lines:

```
DELETE/2=IGNORE  ?FILE.TEMP
RENAME %2%  ?FILE.TEMP
RENAME %1%  %2%
RENAME  ?FILE.TEMP  %1%
```

The first command ensures that file ?FILE.TEMP doesn't exist prior to the RENAME commands and that the CLI will not display a message if it tries to delete ?FILE.TEMP when it doesn't exist. If you want to use this macro to switch the names of FILEA and FILEB, give the following command to the CLI.

```
) SWITCH FILEA FILEB ↵
```

In this instance, the CLI substitutes actual argument values for the dummy arguments that appear in the macro statements:

%1% becomes FILEA (the first argument).

%2% becomes FILEB (the second argument).

So, the CLI executes the following commands in response to your macro call:

```
DELETE/2=IGNORE  ?FILE.TEMP
RENAME  FILEB  ?FILE.TEMP
RENAME  FILEA  FILEB
RENAME  ?FILE.TEMP  FILEA
```

The original FILEA is now named FILEB, and the original FILEB is now called FILEA.

If a macro contains a dummy argument, but you do not supply an actual argument for that dummy argument when you call the macro, the CLI uses a null value for the dummy argument. This in itself will not cause an error. If, however, a command within the macro requires that argument, you may receive an error message.

For example, if you had called the SWITCH macro with only one argument,

```
) SWITCH FILEA ↵
```

the value of %2% would be null. As a result, you would receive an error on the first RENAME command, because it requires two arguments — and you only gave it one argument.

A Programming Analogy

If you know a high–level programming language, this section may interest you.

Creating a macro with simple arguments is similar to creating a subprogram in a high–level language. In both cases you work with dummy arguments whose values are unknown to the macro/subprogram until the CLI/main program invokes/calls the macro/subprogram. For example, compare the following CLI macro and FORTRAN 77 subroutine subprogram line by line. (A FORTRAN 77 statement that begins with the letter C is a comment; an exclamation point — ! — also starts a comment.)

CLI Command

```
COMMENT Macro ADD_2_INTEGERS
COMMENT   to add 2 unsigned integers.
VAR0 [!UADD %1% %2%]
WRITE The sum is [!VAR0]
COMMENT End of the macro.
```

The following CLI command invokes macro ADD_2_INTEGERS.

```
ADD_2_INTEGERS 138 40
```

and the following text appears on the screen.

The sum is 178

FORTRAN 77 Statement

```
C  Subprogram ADD_2_INTEGERS
C     to add 2 integers.
      SUBROUTINE ADD_2_INTEGERS
     +   (FIRST, SECOND)
      INTEGER FIRST, SECOND, VAR0
C
      VAR0 = FIRST + SECOND
      PRINT *, "The sum is ", VAR0
      RETURN  ! to the main program.
```

After the main program has been compiled, linked, and executed, the following FORTRAN 77 statement calls subprogram ADD_2_INTEGERS

```
CALL ADD_2_INTEGERS (138, 40)
```

and the following text appears on the screen.

The sum is 178

The key statement in the macro ADD_2_INTEGERS is

```
VAR0 [!UADD %1% %2%]
```

The CLI responds to it by

1. Replacing the first dummy argument (%1%) by the first actual argument from the command ADD_2_INTEGERS 138 40; this value is 138.
2. Replacing the second dummy argument (%2%) by the second actual argument from the command ADD_2_INTEGERS 138 40; this value is 40.
3. Performing an unsigned addition (!UADD) of the first actual argument and the second actual argument. The sum goes into the CLI variable whose name is VAR0. Chapter 5 explains the [!UADD] pseudomacro.

The next section describes how you can use a dummy argument to specify a range of actual arguments.

Specifying a Range of Arguments

Often you will want a macro statement to refer to several actual arguments. The macro statement could then specify several single dummy arguments. For example

```
DELETE %2% %3% %4%
```

would delete the files specified as the second, third, and fourth arguments in the CLI command that invoked the macro.

A *range dummy argument,* however, lets you specify more than one actual argument with just one dummy argument. You specify the first and last arguments in the range (each identified by its numeric position) and an increment value for the range. The format is

```
%m-n,i%
```

where

m is the first argument in the range.

n is the last argument in the range.

i is the increment value.

Use a hyphen to separate the beginning and end of the range; if you specify an increment, precede it with a comma.

Each number in a range dummy argument is optional. You can, for example, omit the increment value, the beginning of the range, or the end of the range. The CLI uses the following values for omitted parts of a dummy range argument:

If you omit

m the range begins with argument 1.

n the range ends with the last argument supplied (the CLI uses the value 32767).

i the increment value is 1 (which includes every argument in the specified range).

Here are some examples.

Command	Expands to	
DELETE %2–4,1%	DELETE arg2 arg3 arg4	(Starts at 2, with increment of 1.)
DELETE %2–4,2%	DELETE arg2 arg4	(Starts at 2, with increment of 2.)
DELETE %–4,2%	DELETE arg1 arg3	(Starts at 1, with increment of 2.)
DELETE %2–,2%	DELETE arg1 arg33	(Starts at 1, with increment of 2.)

Because you can omit any or all of these range elements, there are several formats you can use to specify a dummy range argument. Table 4–1 explains each format.

Table 4–1 Dummy Range Argument Formats

Range Argument Format		Expands To
%m–n,i%		Argument m and every ith argument through argument n.
%m–n%	(i omitted)	Argument m through argument n.
%m–,i%	(n omitted)	Argument m and every ith argument through the last argument given.
%m–%	(n,i omitted)	Argument m through the last argument given.
%–n,i%	(m omitted)	Argument 1 and every ith argument through argument n.
%–n%	(m,i omitted)	Argument 1 through argument n.
%–,i%	(m,n omitted)	Argument 1 and every ith argument thereafter.
%–%	(m,n,i omitted)	Every argument from the first through the last — that is, every argument.

Notice that the hyphen is required for all range dummy arguments; if you include an increment value, you must precede it with a comma.

For example,

%2–4% specifies the second, third, and fourth arguments.

%2–4,2% specifies only the second and fourth arguments.

%1–10,3% specifies the first, fourth, seventh, and tenth arguments.

%4–11,3% specifies the fourth, seventh, and tenth arguments.

%4–12,3% specifies the fourth, seventh, and tenth arguments.

%4–13,3% specifies the fourth, seventh, tenth, and thirteenth arguments.

Hence, %1–4,1%, %1–4%, and %–4% are all equivalent. If you call the macro with only four arguments, then %–% also expands to the first, second, third, and fourth arguments.

You can preview the effects of a macro that contains dummy range arguments. To do this, place WRITE commands at the beginning of key statements in your macro, verify that they give the desired results, and then remove the WRITE commands. Macro RANGE_DUMMY_EXAMPLE.CLI, following, shows this technique. This macro includes the previous six examples as statements 6 through 11.

```
COMMENT This is macro RANGE_DUMMY_EXAMPLE.CLI.
WRITE TYPE 1, %1%
WRITE TYPE 2, %1%, %2%
WRITE TYPE 3, %1–3%
WRITE TYPE 4, %1–3,2%
WRITE TYPE 5, %2–%
WRITE TYPE 6, %2–4%
WRITE TYPE 7, %2–4,2%
WRITE TYPE 8, %1–10,3%
WRITE TYPE 9, %4–11,3%
WRITE TYPE 10, %4–12,3%
WRITE TYPE 11, %4–13,3%
```

```
) RANGE_DUMMY_EXAMPLE A B C D E F G H I J K L M N O ↵
```

```
TYPE 1 A
TYPE 2 A B
TYPE 3 A B C
TYPE 4 A C
TYPE 5 B C D E F G H I J K L M N O
TYPE 6 B C D
TYPE 7 B D
TYPE 8 A D G J
TYPE 9 D G J
TYPE 10 D G J
TYPE 11 D G J M
```

Range Argument Examples

Suppose the macro EMPTY.CLI contains these two lines:

```
DELETE/2=IGNORE (%–%)
CREATE (%–%)
```

If you type

```
) EMPTY FILEA FILEB FILEC ↵
```

the CLI executes the following commands.

```
DELETE/2=IGNORE (FILEA FILEB FILEC)
CREATE (FILEA FILEB FILEC)
```

The result is three empty files and no error messages if files FILEA, FILEB, and FILEC do not exist. (If the /2=IGNORE switch were not present and the three files did not exist, the CLI would return error messages about nonexistent files and stop executing the macro.)

Or suppose you have the macro RNAM.CLI that contains this command:

```
RENAME (%1-,2%) (%2-,2%)
```

This renames all odd–numbered arguments to their even–numbered counterparts. For instance, the command

```
) RNAM A B C D E F ↵
```

expands to

```
RENAME A B
RENAME C D
RENAME E F
```

You can verify this answer via the following initial version of RNAM.CLI.

```
COMMENT This is macro RNAM.CLI.
WRITE RENAME (%1-,2%) (%2-,2%)
```

Test by typing

```
) RNAM A B C D E F ↵
```

RENAME A B
RENAME C D
RENAME E F

Specifying Switches

By default, if an actual argument includes one or more switches, the corresponding dummy argument returns the argument value with its switches. Thus, if the first argument in a macro call is MYFILE/L, the CLI replaces dummy argument %1% with MYFILE/L.

You can use the slash (/) and backslash (\) characters with dummy arguments to include or exclude specific switch information. Table 4–2 shows formats for dummy arguments and switches..

Table 4–2 Formats of Dummy Arguments and Switches

Dummy Argument	Meaning
%0%	The macro name (with all its switches).
%n%	The nth argument (with all its switches) where n is the numeric position of the argument.
%n/%	The switches on argument n.
%n\%	Argument n without its switches.
%n/switch%	The specified switch(es) on argument n, including any value specified for the switch(es).
%n/switch=%	The value supplied for the specified switch on argument n.
%n\switch%	All of the nth argument's switches except the ones specified.

NOTE: You can omit n from a switch dummy argument when you refer to argument 0 (the macro itself). For example, the dummy argument %/switch=% is equivalent to %0/switch=%.

To retrieve an argument's switches without the argument itself, follow the argument number with a slash. For example, the dummy argument

%1/%

resolves to all the switches appended to argument 1. If argument 1 has no switches, the dummy argument resolves to a null value.

Similarly, to exclude all the switches for a specific argument, follow the argument number with a backslash. The next dummy argument resolves to the value of argument 2 without any of the switches attached to it.

%2\%

To retrieve a specific switch, append that switch to the dummy argument. For example, the dummy argument

%1/S%

resolves to /S if that switch is appended with argument 1; otherwise, the dummy argument resolves to a null value.

If the /S switch includes a value (such as /S=5), the dummy argument resolves to /S=5. You could extract the switch value alone by using the following dummy argument instead.

%1/S=%

If argument 1 includes the switch /S=5, the dummy argument then resolves to the value 5. (If /S appears without a value or the switch does not appear at all, the dummy argument returns a null value.)

You can use a backslash to exclude a specific switch. For example, the dummy argument

%2\L%

resolves to all switches (if any) appended to argument 2, *except* /L. If argument 2 includes only /L or no switches at all, the dummy argument resolves to a null value.

Example with Switches and Dummy Arguments

The following macro, SWITCHES.CLI illustrates the use of several switch dummy arguments. The macro contains these lines.

```
COMMENT This is macro SWITCHES.CLI.
WRITE The switches on the macro name are %0/%
WRITE The switches on the first argument are %1/%
WRITE The value of the /L= switch is %0/L=%
WRITE The macro was called with the following switches
WRITE not including /S and /T: %0\S\T%
WRITE The first argument without its switches is %1\%
```

Call the macro.

```
) SWITCHES/L=7/S/T  FILEA/D/R ↵
```

The switches on the macro name are /L=7/S/T
The switches on the first argument are /D/R
The value of the /L= switch is 7
The macro was called with the following switches
not including /S and /T: /L=7
The first argument without its switches is FILEA

Using Parentheses and Brackets in Macros

Parentheses in macro calls work just as they do in commands. That is, they cause the macro to repeat once for each item enclosed. If you group macro arguments within parentheses, the CLI treats the group as a single argument when resolving dummy arguments; this is sometimes called *indirection*. Thus, you can use parentheses to supply more than one value for a single dummy argument.

For example, suppose the macro ASM.CLI contains the following lines.

```
XEQ MASM/L=%1%.LS %1%
QPRINT %1%.LS
```

If you call the macro with the command

```
) ASM (FILEA,FILEB,FILEC) ↵
```

the CLI expands the command to

```
ASM FILEA
ASM FILEB
ASM FILEC
```

which results in the following sequence of commands:

```
XEQ MASM/L=FILEA.LS FILEA
QPRINT FILEA.LS

XEQ MASM/L=FILEB.LS FILEB
QPRINT FILEB.LS

XEQ MASM/L=FILEC.LS FILEC
QPRINT FILEC.LS
```

The dummy argument %1% in ASM.CLI takes only one actual argument, but the parentheses in the command line cause the macro to execute three times. Each time, one of the arguments enclosed in parentheses in the command line is used for dummy argument %1%.

Using Parentheses with the !FILENAMES Pseudomacro

Occasionally, you will want to work with a group or sequence of filenames at once. By enclosing individual !FILENAMES pseudomacro arguments within parentheses, you can supply multiple file lists for the operation to be executed.

Suppose you have a macro that converts your current year's work for the start of a new calendar year. You want to re–use the files in a particular directory from the previous year, but you want the filenames to reflect the current year. You can rename all files that end in "91" so that they end in "92" instead, using the following command line within the macro:

```
RENAME ([!FILENAMES +91]) ([!FILENAMES +92])
```

Or, suppose you need to show the length of every file in a particular directory; you could accomplish this using a command line in your macro such as this:

```
WRITE  ([!SIZE ([!FILENAMES +])])
```

This technique is useful with any command that has restrictions on the the number of arguments you can supply. See the RENAME and TAR_VS command descriptions in Chapter 5 for other practical examples of how !FILENAMES works in a parenthetical construction.

Using Square Brackets to Specify a File's Contents

When you surround a filename with square brackets, the CLI uses the *contents* of the file rather than the filename.

When you type

```
) TDP ↵
```

the CLI looks for the file TDP.CLI (or TDP, then TDP.PR if it does not find TDP.CLI), and tries to execute the contents of that file. Here you are implicitly calling the macro. The CLI will respond identically if you type

```
) [TDP] ↵
```

The bracket syntax enables you to execute a macro that has the same name as a CLI command (or an abbreviation of the command name). If you try to execute a macro called TI.CLI by typing

```
) TI ↵
```

the CLI executes the TIME command (because it looks for a matching CLI command before it looks for a matching macro). If, however, you type

```
) [TI] ↵
```

the CLI knows you are referring to a file (not a command) and will execute TI.CLI.

With brackets, moreover, you can use the *contents* of a file as an argument to a macro or command. Suppose a file called MYFILES contained the following text.

```
FILEA,FILEB,FILEC,FILED
```

If you type

```
) QPRINT  [MYFILES] ↵
```

the CLI expands the command line to

```
QPRINT FILEA,FILEB,FILEC,FILED
```

and therefore outputs the four files to the line printer.

Overriding Text Line Delimiters

When creating a text file with certain editors or the CREATE/I command, you must end each line of the line with a NEW LINE or other delimiter character. If you use the contents of such a file as an argument to a CLI command, the CLI interprets each delimiter as the end of a command.

For example, if the file PRINTFILES contains these lines

```
FILEA
FILEB
FILEC
```

and you issue the command

```
) QPRINT [PRINTFILES] ↵
```

the CLI uses the contents of PRINTFILES, and generates the commands

```
QPRINT FILEA
FILEB
FILEC
```

Unless the names FILEB and FILEC refer to macro files, the second and third commands will cause an error.

If you want the CLI to ignore line delimiters, type an ampersand (&) immediately before pressing the NEW LINE key. The ampersand causes the CLI to ignore the delimiter and continue with the next line of text.

If the file PRINTFILES contained these lines

```
FILEA &
FILEB &
FILEC
```

the CLI would interpret the command

```
) QPRINT [PRINTFILES] ↵
```

as

```
QPRINT FILEA FILEB FILEC
```

even though the last character of each line in PRINTFILES is a delimiter.

Using Conditional Pseudomacros

The CLI provides several pseudomacros that you can use to execute a series of CLI commands if a specified condition is either true or false. For this reason we call them *conditional* pseudomacros. Programmers will recognize conditional pseudomacros as comparable to IF/THEN/ELSE statements used in high–level programming languages.

Every conditional pseudomacro begins a series of statements that ends with an !END pseudomacro. The series can include an !ELSE pseudomacro. The basic format follows.

```
conditional–pseudomacro
        ...
        (CLI commands to be executed only if the condition is true)
        ...
[!ELSE]
        ...
        (CLI commands to be executed only if the condition is false)
        ...
[!END]
```

If the series *does not* include !ELSE, and the condition is true, the CLI executes all the commands in the series. But if the condition is false, the CLI skips the commands in the series and continues execution with the command (if any) that follows the !END statement.

If the series *does* include an !ELSE statement, the CLI executes the command(s) up to the !ELSE if the condition is true. But if the condition is false, the CLI executes the command(s) that follow !ELSE.

You can improve the readability of a macros that contains conditional pseudomacros by indenting the commands following the conditional pseudomacros and the [!ELSE] statements. One tab character is usually enough of an indentation.

Each conditional pseudomacro takes exactly two arguments. A comparison of the arguments results in either a true or false condition. You can use spaces or commas to separate arguments. If an argument is null, however, you must use a comma. For example, both

```
[!EQUAL,,%1%]
```

and

```
[!EQUAL %1%, ]
```

determine whether or not the first dummy argument is null.

Table 4–3 summarizes the conditional pseudomacros that the CLI provides. For detailed information about these pseudomacros, see Chapter 5.

Table 4–3 Conditional Pseudomacros

Pseudomacro	Condition Tested
!EQUAL	The two arguments are equal (when compared character by character).
!NEQUAL	The two arguments are not equal (when compared character by character).
!UEQ	The two integer arguments (or !VAR/!VARn values) are equal in value.
!UNE	The two integer arguments (or !VAR/!VARn values) are not equal in value.
!ULT	The first integer argument (or !VAR/!VARn value) is less than the second.
!ULE	The first integer argument (or !VAR/!VARn value) is less than or equal to the second.
!UGT	The first integer argument (or !VAR/!VARn value) is greater than the second.
!UGE	The first integer argument (or !VAR/!VARn value) is greater than or equal to the second.

The !EQUAL and !NEQUAL pseudomacros accept any strings as valid arguments: numbers, literal words, macros, or pseudomacros. (To treat them as a single argument, surround multiple words or arguments with parentheses.)

The CLI treats the arguments that you pass to !EQUAL and !NEQUAL as strings and compares them character by character. If one string is longer than another, the strings are unequal. The CLI considers uppercase and lowercase letters to be equivalent.

The other pseudomacros require both arguments to be unsigned decimal integers.

As an example, the following macro determines whether a system operator is on or off duty. If so, the macro queues the arguments given to it as a batch job.

```
COMMENT This is macro BATCH_CHECK_FOR_OP.CLI.
[!equal,[!operator],on]
        write An operator is on duty — will run your batch job.
        qbatch %-%
[!else]
        write An operator is not on duty. Try later.
[!end]
```

The first argument, [!OPERATOR], is a pseudomacro that returns a value of either ON or OFF (as explained in Chapter 5). The CLI compares this result character by character with the second argument, ON. So, if !OPERATOR returns the value ON, the CLI executes the QBATCH command; otherwise, the CLI prints a message to say that the operator is off duty.

Note that you must include separators between the pseudomacro and the first argument, and between the first and second arguments. The indentations on the third and fifth lines are not necessary for the proper execution of the macro. Their value is that they make the macro much easier to read and understand. The CLI ignores any spaces or tabs at the beginning of a line.

Creating a Loop with Conditional Pseudomacros

The !EQUAL and !NEQUAL pseudomacros let you simulate a loop by testing the value of an incrementing variable and calling the macro again (recursively) until the variable matches a predefined value. This section shows an example.

In this example, the conditional pseudomacro !ULE creates a loop. The macro takes the following arguments:

- A CLI command or macro to be executed repeatedly
- A starting index value
- An ending index value
- An increment value

The macro executes the specified command or macro (argument 1), and then increments the starting value (argument 2) by the increment value (argument 4). This process continues until the index value exceeds the ending value (argument 3).

The body of the macro, REPEAT.CLI, is

```
COMMENT This is macro REPEAT.CLI.
[!ULE %2% %3%]
        WRITE Continue looping – Index variable is %2%
        COMMENT Now execute the command given as the first argument.
        %1%
        COMMENT Now call the macro again while incrementing argument 2.
        %0%  %1% [!uadd %2% %4%] %3% %4%
[!ELSE]
        WRITE Stop looping – Index variable is %2%
[!END]
```

This macro is recursive, that is, it calls itself to simulate a loop. The two WRITE statements report the changing index variable as the macro executes each loop.

NOTE: In order to make efficient use of space, a recursive macro call should be the last command in the macro. There should be no further commands after the last [!END] statement.

We can demonstrate this macro by having it execute another simple macro, TEST.CLI, which contains a single command line:

```
WRITE —— Executing the macro TEST.CLI ——
```

The following command calls the REPEAT macro, and specifies TEST.CLI as the first argument, 2 as the starting index value, 10 as the ending index value, and 3 as the increment value:

```
) REPEAT TEST 2 10 3 ↵
```

Continue looping – Index variable is 2
—— Executing the macro TEST.CLI ——
Continue looping – Index variable is 5
—— Executing the macro TEST.CLI ——
Continue looping – Index variable is 8
—— Executing the macro TEST.CLI ——
Stop looping – Index variable is 11
)

Nesting Conditional Pseudomacros

The series of statements belonging to a conditional pseudomacro can contain other conditional pseudomacros. This is called *nesting*. You can nest conditional pseudomacros as deep as you want (subject only to the limit of available memory).

```
conditional–pseudomacro
...
...
        conditional–pseudomacro
        ...
        ...
        [!END]
...
...
[!END]
```

Nested conditional pseudomacros cannot overlap. Each must be entirely enclosed within the larger conditional series. In macros that use nested conditional pseudomacros, the relationship between each conditional pseudomacro and the corresponding !ELSE and !END statements is often difficult to see at a glance. Good macro writers almost always use increasing levels of indentation to align related statements. (See the CALCULATOR.CLI macro example at the end of this chapter.)

If the macro contains too many !END statements, the CLI executes the macro up to the first !END and returns an error message. If the macro does not contain an !END statement for each conditional pseudomacro, the CLI executes all commands up to the pseudomacro that lacks a corresponding !END statement, and then it displays a special prompt (as described in the next section).

If an !ELSE statement falls outside a conditional series, the CLI issues an error message. If a conditional series includes too many !ELSE statements, the CLI executes macro statements up to the conditional pseudomacro that contains the extra !ELSE statement(s), and then it displays an error message.

Terminating a Conditional Pseudomacro

An !END pseudomacro must complete the series of statements that a conditional pseudomacro began. If you execute a macro that does not have an !END statement for each conditional pseudomacro, the CLI issues one of two prompts to indicate that the macro cannot complete execution until you supply an !END statement.

CLI Prompt	Signifies
\)	A conditional pseudomacro lacking an !END statement has tested false; or the series included an !ELSE pseudomacro and the conditional tested true.
!)	A conditional pseudomacro lacking an !END statement has tested true; or the series included an !ELSE pseudomacro, and the conditional tested false.

When you receive one of these prompts, you are in the midst of the macro execution. If you want to abort the macro execution, type a console interrupt sequence (CTRL–C CTRL–A). Or you can resume the macro execution by typing [!END]. Examples are

!) [!END] ↵

and

\) [!END] ↵

The macro resumes execution after you type [!END]. If another such prompt appears after you type [!END], yet another conditional pseudomacro lacks an !END statement. Type [!END] again to close that series.

NOTE: If you change the CLI prompt (prefix string) via the PREFIX command (described in Chapter 5), the ! or \ symbol precedes the new prompt.

If a complex macro contains one or more incomplete conditional series, you may not know which conditional pseudomacro lacks an !END statement. You can use the TRACE command (described in Chapter 5) with the /PSEUDO switch to have the CLI display each pseudomacro before executing it. This may help you isolate the incomplete conditional series.

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.

Table 4–5 CALC.CLI Macro Syntax

Macro–Switch Format	Result
CALC i j *[...]*	Adds j, i, and any additional arguments, and displays the result. To add 3, 989, and 51, for example:) CALCULATOR 3 989 51 ↵ *Adding 3 989 51. Sum is 1043.*
CALC/S i j *[...]*	Subtracts j and any additional arguments from i, and displays the result. To subtract 10 from 20, for example:) CALCULATOR/S 20 10 ↵ *Subtracting from 20 sum of 10. Result is 10.*
CALC/M i j *[...]*	Multiplies i by j, and any additional arguments, and displays the result. To multiply 10 by 35, for example:) CALCULATOR/M 10 35 ↵ *Multiplying the following: 10 35. Product is 350.*
CALC/D i j *[...]*	Divides i by j and any additional arguments, and displays the result. To divide 29 by 3, for example:) CALCULATOR/D 29 3 ↵ *Dividing 29 by the following: 3. Quotient is 9 and remainder is 2.*
CALC/O i *[...]*	Converts i and any additional arguments from a decimal (base 10) value to an octal (base 8) value, and displays the result. To convert 16 base 10 to base 8, for example:) CALCULATOR/O 16 ↵ *Octal value of decimal 16 is 20.*
CALC/DEC i *[...]*	Converts i and any additional arguments from an octal (base 8) value to a decimal (base 10) value, and displays the result. To convert 16 base 8 to base 10, for example:) CALCULATOR/DEC 16 ↵ *Decimal value of octal 16 is 14.*

```
COMMENT This is macro CALC.CLI.

[!EQUAL,(%1%),()]
  COMMENT Display the following lines if no arguments are given.
  WRITE This macro does arithmetic operations using integer arguments.
  WRITE Use /A to add two or more integers - CALC/A 45 89 5
  WRITE Use /S to subtract one integer from another - CALC/S 39 22
  WRITE Use /M to multiply two integers - CALC/M  45 37
  WRITE Use /D to divide two integers - CALC/D  400 26
  WRITE Use /O to change a decimal number to octal - CALC/O 29
  WRITE Use /DEC to change an octal number to decimal - CALC/DEC 44

[!ELSE]
  COMMENT Arguments were given.  Check switch, then do calculation.
  [!EQUAL,%0/%,]
    WRITE
    WRITE Adding %1-%. Sum is [!UADD,%1-%].
    WRITE

  [!ELSE]
    COMMENT There is a switch - check for S.
    [!EQUAL,%0/%,/S]
      WRITE
      WRITE Subtracting from %1% sum of %2-%. Result is &
      [!USUBTRACT,%1-%].
      WRITE

    [!ELSE]
      COMMENT The switch is not S - check for M.
      [!EQUAL,%0/%,/M]
        WRITE
        WRITE Multiplying the following: %1-%. Product is &
        [!UMULTIPLY,%1-%].
        WRITE

      [!ELSE]
        COMMENT The switch is not S or M - check for D.
        [!EQUAL,%0/%,/D]
          WRITE
          WRITE Dividing %1% by the following: %2-%.Quotient is &
          [!UDIVIDE,%1-%] and remainder is [!UMODULO,%1-%].
          WRITE
```

Figure 4–1 The Calculator Macro (continues)

```
        [!ELSE]
          COMMENT The switch is not S, D, or M - check for O.
          [!EQUAL,%0/%,/O]
            WRITE
            WRITE Octal value of decimal %1-% is [!OCTAL,%1-%].
            WRITE
          [!ELSE]
            COMMENT The switch is not S, D, M, or O-check for DEC.
            [!EQUAL,%0/%,/DEC]
              WRITE
              WRITE Decimal value of octal %1-% is [!DECIMAL,%1-%].
              WRITE

            [!ELSE]
              COMMENT The switch is not null, S, D, M, O, or DEC.
              WRITE
              WRITE Illegal switch - no calculations done.
              WRITE

              COMMENT Each of the next seven !ENDs closes an !EQUAL.
              COMMENT

            COMMENT The next !END closes /DEC processing.
            [!END]

          COMMENT The next !END closes /O processing.
          [!END]

        COMMENT The next !END closes /D processing.
        [!END]

      COMMENT The next !END closes /M processing.
      [!END]

    COMMENT The next !END closes /S processing.
    [!END]

  COMMENT The next !END closes /A processing.
  [!END]

COMMENT The next !END closes the processing done with no arguments.
[!END]
```

Figure 4–1 The Calculator Macro (concluded)

Macro SPACEX.CLI

Macro SPACEX.CLI checks the amount of disk space available in your initial user directory (:UDD:username) and three logical disk units (LDUs): the root directory (:), :UDD1, and :CEO. If your system does not include LDUs with these names, you can adapt the macro to use the LDUs you do have. Just adapt the macro sections that use LDU names to reflect each LDU on your system.

```
comment Macro SPACEX.CLI, computes disk space as a percentage in
comment one's user directory and three LDU directories
comment - :, :UDD1, and :CEO.
space/str=user_space :udd:[!username]
space/str=root_space :
space/str=udd1_space :udd1
space/str=ceo_space :ceo
space/str=backups1_space :backups1
string [!string/name=user_space]

comment Get the second value and fourth values returned by the
comment SPACE command.  These are the total amount of disk space
comment and the amount currently used.  Put the two values
comment in variables.
var/name=total [!argument/item=2 [!string]]
var/name=used [!argument/item=4 [!string]]
comment Multiply the amount of space used by 1000 -- to yield
comment three digits of percentage figure -- before dividing.
var/name=used000 [!umul [!var/name=used],1000]
comment Divide amount of space used - multiplied by 1000 -
comment by total amount to get the percentage number.
var/name=percent0 [!udiv [!var/name=used000] [!var/name=total]]
comment Format the percentage figure as a string.
var/name=percent [!udiv [!var/name=percent0],10]
var/name=tenths [!umod [!var/name=percent0],10]
str/name=tmp5 [!var/name=percent].[!var/name=tenths]
comment Format the information for display.
str/name=user [!user][!asc 11][!asc 11][!str/name=user_space].,&
,Used [!str/name=tmp5]%
```

Figure 4–2 Macro to Display Disk Usage as a Percentage (continues)

```
comment Repeat for root directory.
string [!string/name=root_space]
var/name=total [!argument/item=2 [!string]]
var/name=used [!argument/item=4 [!string]]
var/name=used000 [!umul [!var/name=used],1000]
var/name=percent0 [!udiv [!var/name=used000] [!var/name=total]]
var/name=percent [!udiv [!var/name=percent0],10]
var/name=tenths [!umod [!var/name=percent0],10]
str/name=tmp5 [!var/name=percent].[!var/name=tenths]
comment Format the information for display.
str/name=root :[!asc 11][!asc 11][!str/name=root_space].,,Used &
        [!str/name=tmp5]%

comment Repeat for UDD1.
string [!string/name=udd1_space]
var/name=total [!argument/item=2 [!string]]
var/name=used [!argument/item=4 [!string]]
var/name=used000 [!umul [!var/name=used],1000]
var/name=percent0 [!udiv [!var/name=used000] [!var/name=total]]
var/name=percent [!udiv [!var/name=percent0],10]
var/name=tenths [!umod [!var/name=percent0],10]
str/name=tmp5 [!var/name=percent].[!var/name=tenths]
comment Format the information for display.
str/name=udd1 UDD1[!asc 11][!asc 11][!str/name=udd1_space].,,Used &
        [!str/name=tmp5]%

comment Repeat for CEO.
string [!string/name=ceo_space]
var/name=total [!argument/item=2 [!string]]
var/name=used [!argument/item=4 [!string]]
var/name=used000 [!umul [!var/name=used],1000]
var/name=percent0 [!udiv [!var/name=used000] [!var/name=total]]
var/name=percent [!udiv [!var/name=percent0],10]
var/name=tenths [!umod [!var/name=percent0],10]
str/name=tmp5 [!var/name=percent].[!var/name=tenths]
comment Format the information for display.
str/name=ceo CEO[!asc 11][!asc 11][!str/name=ceo_space].,,Used &
        [!str/name=tmp5]%

write
write Space usage for your user directory and all the LDUs is
write [!string/name=user]
write [!str/name=root]
write [!str/name=udd1]
write [!str/name=ceo]
```

Figure 4–2 Macro to Display Disk Usage as a Percentage (concluded)

End of Chapter

Chapter 5
Dictionary of CLI Commands, Macros, Pseudomacros, and Utility Programs

This chapter presents descriptions of CLI commands, macros. pseudomacros, and utility programs in alphabetical order. The major sections proceed as follows.

- Pseudomacros
- Selecting Files by Date and Time
- Universal Command Switches
- Common Questions
- Getting Help
- Summary of CLI Commands, Pseudomacros, Macros, and Utility Programs
- Command Descriptions (ACL through XEQ)

This chapter describes only those utility programs that are not explained in their own manual or in the manual *Managing AOS/VS and AOS/VS II*. For example, the SED text editor has its own manual, thus is not covered here. And the PED and PREDITOR utilities, since they relate to system management, are described in *Managing AOS/VS and AOS/VS II*; they, too, are not described here. The Preface lists the utilities not explained here and the pertinent manuals.

Before you proceed, you may want to review the notation conventions used in command formats. In the command formats, italic square brackets *[]* denote optional entries. When you enter a command, *do not* type these brackets around the optional item. If the brackets are not shown in italic type, however, you must type them, as, for example, in [!ACL].

As an example of a command with an optional entry, the format of the DELETE command is

DELETE pathname *[pathname ...]*

This format says you must have at least one argument. A specific example of the command is

```
) DELETE  MYFILE1 MYFILE2  MYFILE3 ↵
```

This command includes the mandatory single argument (MYFILE1) and optional arguments (MYFILE2 and MYFILE3) which you do not enclose in brackets.

In the examples with dialog between a user and the CLI, all user input lines begin with the CLI prompt and end with a NEW LINE character, as in

```
) TIME ↵
```

Pseudomacros

Some CLI pseudomacros, such as !SEARCHLIST and !DATE, return environmental settings or system variables. Others, like !EQUAL, allow you to specify conditional execution of commands. Still others, like !OCTAL, convert values. The purpose of all macros is to return a value from the system or from your variables. An example of the former is !TIME to return the time of day from the system. An example of the latter is !VAR0 to return the value of one of your 10 variables (!VAR0 through !VAR9).

This manual often says that a pseudomacro "expands to ...". This is just another way of saying that the pseudomacro returns a value. For example, it is equally correct to say that the !TIME pseudomacro expands to the current system time or that the !TIME pseudomacro returns the current system time.

Although this chapter sometimes refers to pseudomacros without their surrounding brackets ([ ]), you must include these brackets in a CLI command that contains the pseudomacro. For example, it's correct to say that the !DATE pseudomacro returns the current date from the operating system. A CLI command to obtain the current date via this pseudomacro must use brackets. An example is

```
) WRITE The current date is [!DATE] ↵
```

Refer to Chapter 4 for information about using pseudomacros for a variety of macro–writing techniques.

Selecting Files by Date and Time

Many commands allow you to select files according to date and time they were created, modified, or last accessed. For example, suppose you have created file MYFILE on January 10, 1990 at 10:14:28 am. The operating system assigns this date and time as the file's time–last–accessed value. If you look at this file later in the day with a text editor, this time value changes according to the time the text editor opened the file.

Suppose you want to obtain basic information about all files in the working directory that you gained access to at any time on January 10, 1990. You need to select all files whose times last accessed are between midnight on January 10, 1990 and 11:59:59 p.m. on January 10, 1990. The switches for these times are respectively

/AFTER/TLA=10–JAN–90:00:00:00 and /BEFORE/TLA=10–JAN–90:23:59:59

The /AFTER/TLA switches select files whose time last accessed is on or after January 10, 1990 at 00:00:00 a.m. (midnight). The /BEFORE/TLA switches select files whose time last accessed is on or before January 10, 1990 at 11:59:59 p.m.

The primary command and switch to obtain basic information about a file is FILESTATUS/ASSORTMENT. Joining this command and switch with the above pair of switches gives the following correct command.

```
) FILESTATUS/ASSORTMENT/AFTER/TLA=10–JAN–90:00:00:00& ↵
&) /BEFORE/TLA=10–JAN–90:23:59:59 ↵
```

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

ACL *Command*

Displays or sets the access control list for a file.

Format

ACL pathname [*username, access−types [...]*
username[:groupname][,...],access−types [...][1]]

[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.*

ACL (continued)

- Accepts templates (for pathname or username).
- No argument switches.
- Requirement: *Standard.*
- See also: DEFACL (to display or set your default ACL), !ACL (to represent the ACL of a specified file), GROUPLIST (to become the member of a group list), and SUPERUSER (to override access control lists).

Why Use It?

Use this command to allow others access to your files, and to specify what type of access you permit them. For example, you may want several individuals to be able to read a file, but not change it in any way. Or you may allow others to add information to the file, without giving them the power to delete it.

You can also use this command to prevent certain users from gaining any access to certain files.

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	(CLI32 only.) Sets or displays ACLs of files accessed (with /TLA=), created (with /TCR=), or modified (with /TLM=) on or after a given date–time. The date/time switches are further described near the beginning of this chapter in the section "Selecting Files by Date and Time."
/BEFORE	(CLI32 only.) Sets or displays ACLs of files accessed (with /TLA=), created (with /TCR=), or modified (with /TLM=) before a given date–time. The date/time switches are further described near the beginning of this chapter in the section "Selecting Files by Date and Time."
/COUNT	(CLI32 only.) Counts the number of files this command processes.
/D	Assigns the default ACL to the specified file. (Do not specify any username or access codes.)
/DEFAULT	(CLI32 only.) Same as /D.
/K or /KILL	Deletes the file's ACL, allowing only a person with Superuser privilege to access the file. (Do not specify any username or access codes.) Use /KILL for CLI32 only.
/SORT	(CLI32 only.) Sorts alphabetically the filenames this command processes. If you want to see the filenames, you must also use the /V switch.

ACL (continued)

/TOPDOWN — (CLI32 only.) Assigns the ACL from the top of the specified directory tree to the bottom; useful when you are using the # template. If you omit this switch, the system tries to assign the ACL from the bottom up, which would cause the command to fail if you lacked Write access to a lower level directory.

/TRAVERSE=directory–typecode — (CLI32 only.) Specifies the directory types to traverse (go through) while executing this command. Table 2–8 contains valid values of directory typecodes. 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

```
ACL/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 will give expected results.

```
ACL/TRAVERSE=\CPD #:PROJ.–,RE
```

/TYPE=typecode — (CLI32 only.) Specifies one or more types of files to process. Valid type codes appear in Table 2–8.

/V — Shows the filename for each ACL set or changed. (This switch is helpful if you used a pathname template.)

/VERIFY — (CLI32 only.) Same as /V.

ACL Example 1

The following dialog displays the access control list for the file MY_REPORT.

```
) ACL  MY_REPORT ↵
LEE,OWARE  PROJ+,WARE  JONES,RE  J+,
```

The system shows that user LEE has complete access; usernames beginning with PROJ have Write, Append, Read and Execute access; user JONES has Read and Execute access, and other usernames beginning with J are denied access (with a null privilege list). Although JONES matches the template J+, the first match determines the privileges that apply.

To allow user JONES Write and Append access as well, you could type

```
) ACL/V  MY_REPORT  LEE,OWARE  JONES,WARE PROJ+,WARE  J+, ↵
```

Display the ACL for every file in the working directory whose name begins with MY:

```
) ACL/V  MY+ ↵
MYTEST.DATA LEE,OWARE +,WARE
MYPROG.F77 LEE,OWARE JONES,RE
MY_REPORT  LEE,OWARE JONES,WARE PROJ+,WARE J+,
MYPROG.PR LEE,OWARE JONES,RE
```

ACL Example 2

User CHRIS wants to let LEE read the file MYFILE, which resides in a subdirectory called MYDIR in :UDD:CHRIS. First, he checks the file's ACL:

```
) ACL/V  MYFILE⤶
MYFILE  CHRIS,OWARE
```

LEE has no access to the file. To provide access, CHRIS must give LEE Execute access to all user directories above the file:

```
) ACL  :UDD:CHRIS  [!ACL :UDD:CHRIS]  LEE,E ⤶
) ACL  :UDD:CHRIS:MYDIR  [!ACL :UDD:CHRIS:MYDIR]  LEE,E ⤶
```

The pseudomacro !ACL represents the current access control list for the specified file. CHRIS can append information to the existing ACL without having to know or retype it.

Next, CHRIS adds LEE,R to the ACL for MYFILE so that LEE can read the file.

```
) ACL :UDD:CHRIS:MYDIR:MYFILE & ⤶
[!ACL :UDD:CHRIS:MYDIR:MYFILE]  LEE,R ⤶
```

ACL Example 3

```
) ACL/V  :UDD:ALLEN ⤶
:UDD:ALLEN  ALLEN,OWARE  SAM, WARE
) ACL/V  :UDD:ALLEN:REPORTS:#  ALLEN,OWARE  SAM, WARE ⤶
Warning: Owner or write access is required, File 1989:MAY_REPORT

) ACL/V/TOPDOWN  :UDD:ALLEN:REPORTS:#  ALLEN, OWARE  SAM, WARE ⤶
1989:MAY_REPORT
1989:JUNE_REPORT
```

For CLI32 only, this example shows user SAM trying to change the ACL on a directory tree belonging to ALLEN. Since SAM has WARE access to the topmost directory, he should be able to change the ACLs of all subordinate files. But since the system tries to assign ACLs from the bottom up, and SAM lacks Write access to directory REPORTS:1989, the command fails. SAM then retries the ACL command with the /TOPDOWN switch, instructing the system to assign the ACL from top to bottom, and the command works.

ACL Example 4

```
) ACL/V  GROUP_REPORT JKM,OWARE  +:MARK_II,WARE  +,R ⤶
GROUP_REPORT JKM,OWARE +:MARK_II,WARE +,RE
```

For AOS/VS II and CLI32 only, this ACL gives user JKM all access to file GROUP_REPORT; the OWARE specification, since it comes first, overrides any other specification (group or nongroup) for JKM. The ACL gives any user WARE access to the file when he or she is a member of group MARK_II; and it gives all other users Read access to the file.

!ACL *Pseudomacro*

Expands to the access control list for the specified file.

Format

[!ACL pathname]

This pseudomacro returns the access control list for the file specified by pathname; you cannot use a template for pathname.

- No templates.
- No argument switches.
- No macro name switches.
- Requirement: *Standard.*
- See also: ACL (to display or set a file's ACL), DEFACL and !DEFACL (to display or set your default ACL).

Why Use It?

The !ACL pseudomacro is useful as a shorthand method for referring to a file's ACL. You can use the pseudomacro if you do not know a file's ACL, or if you do not want to type the entire ACL into a command line.

One common use (as shown in Example 1) is to use the pseudomacro with the ACL command to append access control information to an existing ACL.

Note, however, that if the new access control information contains a username or template that matches one in the original ACL, the CLI discards the new username or template. For example, if the original ACL contains the template +, that template will match and therefore override any username you could append.

!ACL Example 1

```
) ACL MYFILE [!ACL MYFILE] MARK,WRE )
```

This command changes the access control list for MYFILE by adding MARK,WRE to the file's current ACL. Using the pseudomacro frees you from having to retype the original ACL for the file. If, however, MARK matches any entry in the original ACL (such as MAR+,RE), the new information does not apply because the first matching entry predominates.

!ACL Example 2

```
) ACL YOURFILE [!ACL MYFILE] )
```

This command assigns to YOURFILE the same access control list used by MYFILE. You can use the pseudomacro to duplicate an ACL without knowing its exact contents.

!ARGUMENT *Pseudomacro*

Expands to the requested arguments (CLI32 only).

Format

[!ARGUMENT *[argument ...]*]

This pseudomacro expands to the requested arguments in the command that invoked the pseudomacro. The pseudomacro begins by extracting all the arguments in the command and then expands to only the ones that you requested. For example, suppose that macro SUB1.CLI contains the two statements

```
WRITE All the arguments are [!ARGUMENT %-%]
WRITE The number of arguments from the second through the last is &
    [!ARGUMENT/COUNT/ITEM=2-:1  %-%]
```

and the command line that invokes macro SUB1.CLI is

```
) SUB1 A BC DEFG KLMNOP ↵
```

The first statement in the macro displays all the arguments in the command line; they are A, BC, DEFG, and KLMNOP. The second statement counts, but does not display, the requested arguments in the command line. The exact output from invoking SUB1 in this case follows.

All the arguments are A BC DEFG KLMNOP
The number of arguments from the second through the last is 3

- No templates.
- Accepts macro name switches (described later).
- Requirement: *Standard.*
- See also: the section "Using Dummy Arguments" in Chapter 4.

Why Use It?

Use this pseudomacro to obtain some or all of the arguments that are passed to a macro. You can also easily count these arguments and take action according to whether or not the count is what the macro expects.

Macro Name Switches

/COUNT	Expands to the number of arguments supplied to the pseudomacro and limited by argument-list (from the /ITEM switch). Specifying /COUNT means !ARGUMENT returns only a number and none of the arguments.
/FILL	Expands to null arguments if the requested arguments do not exist. For example, suppose you request seven arguments but only five exist. The last two arguments become null ones.

!ARGUMENT (continued)

/ITEM=argument–list

Specifies the arguments to be extracted. To specify a single item, use an integer that specifies the location of the item in the argument list. For example, the command

```
) WRITE [!ARGUMENT/ITEM=3 abcd xyz pqr mmno] ↵
```

displays "pqr" since it is the third item in the argument list. To specify a range of items, argument–list must have the form m–n:i
where

m	represents the first argument
n	represents the last argument
i	represents the increment value

For example,

2–4:1	specifies arguments 2, 3, and 4.
2–12:3	specifies arguments 2, 5, 8, and 11.

The default values of m, n, and i are as follows.

m	1
n	The number of arguments
i	1

For example,

/ITEM=–12:3	specifies arguments 1, 4, 7, and 11.
/ITEM=3–:2	specifies arguments 3, 5, 7, 9, ...
/ITEM=2–4:	specifies arguments 2, 3, and 4.

The format of argument–list is identical to that of dummy arguments as explained in Chapter 4, except that the colon (:) separates the last argument and increment value specifiers.

To specify multiple lists of arguments, separate each item_list with commas and place parentheses around the multiple occurrences of argument–list. For example,

/ITEM=(4,7–9,3–7:2,11) specifies these arguments:
4th; 7th, 8th, 9th; 3rd, 5th, 7th; 11th

/MACRO

This switch is valid in a macro only. It tells the CLI to use the arguments to the macro arguments, as if you specified [!ARGUMENT %–%] instead of a list of arguments. For example, say you have a macro TEST that contains

```
write [!argument/macro]
```

Call TEST with 0 and 2 arguments:

```
) TEST A B ↵
A B
```

If you use /MACRO outside of a macro, or if you use a pseudomacro as a macro argument, the CLI will ignore it. Used together with the /ITEM switch, you can extract specified arguments supplied by /MACRO.

!ARGUMENT Example 1

Next is macro SUB2.CLI followed by the results of invoking it with eight arguments.

```
comment This is macro SUB2.CLI.
comment From all arguments, display the 3rd, 5th, 7th, etc.
WRITE 1.,,[!ARGUMENT/ITEM=3-:2 %-%]
comment From all arguments, display the count of the 3rd,  5th,
comment    7th, etc.
WRITE 2.,,[!ARGUMENT/COUNT/ITEM=3-:2 %-%]
comment From all the arguments, display the 9th through 12th.
WRITE 3.,,[!ARGUMENT/ITEM=9-12:1 %-%]
comment  From all arguments,display the count of the 9th through
comment  the 12th.
WRITE 4.,,[!ARGUMENT/COUNT/ITEM=9-12:1 %-%]
comment From the arguments, display the 2nd, 5th, 8th, etc.
WRITE 5.,,[!ARGUMENT/ITEM=2-:3 %-%]
comment From the 4th, 5th, 6th, and 7th arguments, display
comment    the 2nd, 5th, 8th, etc.
WRITE 6.,,[!ARGUMENT/ITEM=2-:3 %4-7%]
comment From none of the arguments, display 2nd, 5th, 8th, etc.
comment Note that all arguments are ignored.
WRITE 7.,,[!ARGUMENT/ITEM=2-:3]
comment From all arguments, display all of them.
WRITE 8.,,[!ARGUMENT %-%]
comment From all arguments, display the count of all of them.
WRITE 9.,,[!ARGUMENT/COUNT %-%]
comment End of macro.
```

The following dialog shows what happens when you call SUB2 with eight arguments.

```
) SUB2 a b c d e f g h ↵

1. c e g
2. 3
3.
4. 0
5. b e h
6. e
7.
8. a b c d e f g h
9. 8
```

You can learn about the !ARGUMENT pseudomacro from this example. Modify the arguments in the command that invokes SUB2.CLI, modify the switches /COUNT, and /ITEM=argument-list in SUB2.CLI, or both; then see for yourself what happens as you make these changes.

!ARGUMENT Example 2

Next, along the same lines as the previous example, is macro SUB_MORE.CLI followed by the results of invoking it with eight arguments.

```
comment This is macro SUB_MORE.CLI.
comment The next two statements ignore arguments this macro
comment receives.
comment From 8 arguments S - Z, display the 3rd, 5th, and 7th.
WRITE A.,,[!ARGUMENT/ITEM=3-7:2  S T U V W X Y Z]
comment From 8 arguments S - Z, display the count of the 3rd,
comment    5th, and 7th.
WRITE B.,,[!ARGUMENT/COUNT/ITEM=3-7:2  S T U V W X Y Z]
comment From all arguments, display all of them.  This command
comment and command number 8 in the previous macro are equivalent.
WRITE C.,,[!ARGUMENT/MACRO]
comment From 3 arguments, display the 2nd.
WRITE D.,,[!ARGUMENT/ITEM=2 a b c]
comment From 3 arguments, display the 2nd and then the 1st.
WRITE E.,,[!ARGUMENT/ITEM=2/ITEM=1 a b c]
comment  This is equivalent to [!ARGUMENT/ITEM=1-2:1], so from
comment    3 arguments it will display the 1st and 2nd.
WRITE F.,,[!ARGUMENT/ITEM=-2 a b c]
comment This is equivalent to [!ARGUMENT/ITEM=1-:2], so from
comment    3 arguments it will display the 1st and 3rd.
WRITE G.,,[!ARGUMENT/ITEM=-:2 a b c]
comment From 3 arguments,display the 2nd and then the 1st and 3rd.
WRITE H.,,[!ARGUMENT/ITEM=(2,1-3:2) a b c]
comment  From the three arguments, display the 2nd, then the 5th,
comment   and then the 2nd.  Since the 5th argument doesn't exist,
comment    the /FILL switch creates it as the null argument.
WRITE I.,,[!ARGUMENT/ITEM=(2,5,2)/FILL a b c]
comment End of macro.
```

The following dialog shows what happens when you call SUB2 with eight arguments.

```
) SUB_MORE a b c d e f g h ↵
```

A. U W Y
B. 3
C. a b c d e f g h
D. b
E. b a
F. a b
G. a c
H. b a c
I. b b

!ASCII *Pseudomacro*

Expands to the ASCII character that corresponds to an octal value.

Format

[!ASCII octal–number *[...]*]

This pseudomacro returns the ASCII character that corresponds to the specified octal number. You can include more than one octal number within the pseudomacro.

- No templates.
- No argument switches.
- No macro name switches.
- Requirement: *Standard.*
- See also: WRITE command (includes a table of control characters that you can use with !ASCII)

Why Use It?

You can use !ASCII to include a character that the CLI would normally interpret as a syntax character (such as parentheses, angle brackets, square brackets, ampersand, comma, carriage return, form feed, and NEW LINE). This depends on the country for which your keyboard is designed. Appendix A provides a chart of the ASCII character set.

This pseudomacro is useful if you want to include a non–typeable (non–printable) character in a command. For example, you can use it with the WRITE command to display blinking, underlined, or dim text. You can also use it to sound the terminal's bell tone. (See Example 1.)

!ASCII Example 1

(within a macro)

```
write [!ascii 207] Error!
```

This command causes the terminal to beep and display the message *Error!* Using !ASCII is the only way to include a nonprintable character in a WRITE statement.

!ASCII Example 2

Try the following command on a CRT.

```
) WRITE BRIGHT [!ASCII 240][!ASCII 234] DIM [!ASCII 235] BRIGHT_AGAIN )
BRIGHT  DIM  BRIGHT_AGAIN
```

The [!ASCII 234] changes the display of text from bright to dim and [!ASCII 235] resets the display of text from dim to bright.

ASSIGN *Command*

Assigns a character device for your exclusive use.

Format

ASSIGN devicename *[...]*

This command reserves the named character device(s) for your exclusive use until you either release the device (using the DEASSIGN command) or the current CLI process terminates.

You can assign any character–oriented device (such as a card reader or console) that is not enabled for spooling, and that is not already in use.

- Accepts templates (for device–name).
- No argument switches.
- Requirement: *PID 2.*
- See also: DEASSIGN (to release an assigned device).

Why Use It?

Use this command to dedicate a device exclusively to your CLI process, without risk of losing the resource to another process. (Normally, the EXEC program assigns devices to users when they log on.)

NOTE: After you have finished using it, be sure to release the assigned device so that others can then use it. See the DEASSIGN command.

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=.

ASSIGN Example

```
) ASSIGN @CON5 ↵
...
...
) DEASSIGN @CON5 ↵
```

This command assigns the terminal identified by @CON5 for your exclusive use. After using it, you should issue the DEASSIGN command to release it for others to use.

BIAS *Command*

Displays or sets the system's bias factor.

Format

BIAS *[minimum [maximum]]*

The BIAS command either shows what the current system bias settings are, or lets the system operator (PID 2) set a minimum and maximum bias setting.

The system's bias factor determines the minimum and maximum number of non–interactive, compute–bound processes that the system will try to keep in memory.

The default bias settings are a minimum number of 0 and a maximum number of 32.

- No templates.
- No argument switches.
- Requirement: *Standard* to display; *PID 2* or *System Manager* to set.

Why Use It?

Unless you are the system operator or manager, you won't need this command.

The system operator or system manager can use this command to alter the bias settings to favor interactive or non–interactive processes. Setting the maximum to a lower value, may make it easier for interactive processes to gain CPU time.

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=.

BIAS Example 1

) BIAS ↵
Minimum: 0, Maximum: none

This command displays the current system bias settings: the default settings.

BIAS Example 2

) BIAS 2 ↵

This command sets the minimum bias setting to 2 (that is, the system will try to keep at least two non–interactive processes in memory at all times).

BLOCK *Command*

Suspends a process.

Format

BLOCK { process–ID / processname / username:processname } *[...]*

The BLOCK command blocks (suspends) execution of the specified process. You can specify more than one process with this command. If you use a simple process name, the CLI assumes your username.

You must supply the process ID or the process name. You can use a process name only if the process was created with the PROCESS command and /NAME=name switch.

- No templates.
- No argument switches.
- Requirement: *Standard* to block the CLI or a subordinate process; *PID 2* or *Superprocess* to block any process.
- See also: PROCESS, RUNTIME, TERMINATE.

Why Use It?

Use this command to suspend execution of a process without terminating it. You can block a process to determine whether or not it is having a significant effect on system response time. (The RUNTIME command can tell you how much CPU time a process is using and how much I/O it is doing.)

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=.

BLOCK Example 1

```
) PROCESS/NAME=MYPROG  :UDD:SMITH:PROGA ↵
PID 17

) BLOCK 17 ↵
...
...
) UNBLOCK 17 ↵
```

The first command creates a new process, to which the system assigns PID 17. The next command blocks that process, perhaps to perform related operations. The UNBLOCK command later releases the blocked process. Instead of the PID (17) the person could have used the process name, as in

```
) BLOCK MYPROG ↵
```

BLOCK Example 2

) PROCESS/NAME=JD JMP_DOT ↵ *PID: 31*	(Creates a process with name JD to run program file JMP_DOT.PR.)
) RUNTIME JD ↵ *PID: 31 ...*	(Displays system information about JD.)
) BLOCK JD ↵	(Blocks process JD.)
) TERMINATE JD ↵	(Terminates process JD.)
) CHECKTERMS ↵ *Process Termination, PID: 31* **Abort** *Terminated by a superior process*	(Displays process termination message.)

BRAN *Utility*

Analyzes an AOS/VS or AOS/VS II break file.

Format

XEQ BRAN breakfile–pathname *[symbol–table–pathname]*

When a process terminates abnormally, the system can create a break file, which contains information that can help determine the cause of the termination. The BRAN utility generates a report of the contents of a specified break file. If you also specify a symbol table, the report prints address values symbolically.

The report provides global information about the terminated process, and information about individual tasks. The process information includes the program type, memory usage, the current task (active when the process terminated), and the number of free tasks and active tasks.

For each active task, the report describes the task identifier (task ID), the task's priority, and the values of the program counter, the accumulators, and the stack pointers. In addition, the report lists the system call being serviced at the time of termination.

- Does not accept templates.
- Requirement: *Standard* (E access to program file in :UTIL).
- See also: BREAKFILE.

Why Use It?

Use the BRAN utility if you want specific information about a terminated process that generated a break file. The break file captures details about the state of the process at the time it terminated. The BRAN report presents this information in a format that can either help you determine what caused the process to terminate, or show you what the process was doing at the time. BRAN is designed for programmers who know AOS/VS–AOS/VS II assembly language.

BRAN Switches

/L=pathname	Writes the report to pathname instead of @OUTPUT.

BRAN Example

```
) XEQ BRAN/L=MYPROG.REPORT  ?00086.10_24_45.BRK  MYPROG.ST )
```

This command tells the BRAN utility to analyze the break file ?010.023_026_016.BRK. The command includes the name of the symbol table (MYPROG.ST) so that the listing will display addresses symbolically. The report goes to a file called MYPROG.REPORT.

BREAKFILE *Command*

Displays or sets the break file format for a process.

Format

```
BREAKFILE { process–ID            } [ ... ]
          { processname           }
          { username:processname  }
```

When a process terminates abnormally, the system can create a break file, which contains information that can help determine the cause of the termination. The BREAKFILE command either reports the type of information that will be included in the specified process's break file, or lets you specify the information you want included in the break file.

You must supply the process ID or the process name. You can use a process name only if the process was created with the PROCESS command and /NAME=name switch.

Without switches or arguments, BREAKFILE displays the current break–file status. With either a process–ID (PID) or processname argument and switches, it creates a break–file format for the process specified in the argument. Later, a break file will be created for the process in that format, if any of the following conditions occurs:

- The process traps (encounters an internal error condition).
- The process terminates through a CTRL–C CTRL–B sequence.
- You terminate the process with the TERMINATE command.
- The process terminates through a CTRL–C CTRL–E sequence.

If the process terminates normally, no break file will be created. If you do not specify /ALL, /SHARED, /UNSHARED or /PREAMBLE, the break file's contents will be the preamble of the process.

Break files contain the processing values of a program at the time the program terminates abnormally. If a program terminates normally, the system does not create a break file. Break files are primarily useful to help system programmers identify obscure, persistent program errors.

You can issue a BREAKFILE command on another process located in a ring having equal or greater value than the ring of your current process. You can issue a BREAKFILE command on a subordinate (son) process if you have Superprocess on, or if you have server status and a valid connection to the specified process. If you are a server process, the ring you specify (/RING=n) for the break file must have a value greater than or equal to the connection ring's value. The default ring for user processes is 7.

BREAKFILE (continued)

- No templates.
- No argument switches.
- Requirement: *Standard* if used for the current CLI or for a connected server process; *Superprocess* if used with another process (even a son).
- See also: BRAN (to analyze a break file).

Why Use It?

Use the BREAKFILE command to specify the type of information that you want included in a process's break file whenever the process terminates abnormally. A break file provides diagnostic information, which you can analyze or include with a Software Trouble Report (STR).

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=.

/ALL	Specifies that the break file contains the entire process address space of the selected ring (/RING=n).
/DIRECTORY=pathname	Specifies the directory in which to create the break file.
/FILENAME=pathname	Defines the break file's pathname. The break file name takes the form ?pid.time.ring.BRK.
/KILL	Indicates that the specified process will not create a break file.
/PREAMBLE	Specifies that the break file contains the preamble data only.
/RING=n	Sets the ring number of the specified process's break file where n is a number from 1 to 7. The default ring is 7, which is also the default ring for user programs.
/SHARED	Specifies that the break file contains the shared area only.
/UNSHARED	Specifies that the break file contains the unshared area only.

The following switches are mutually exclusive:

- /ALL, /KILL, /PREAMBLE, /SHARED, and /UNSHARED (If you do not use any of these switches, the CLI assumes /PREAMBLE.)
- /DIRECTORY and /FILENAME (If you do not use either of these switches when setting a break–file format, the system will assign the break file a default name and place it in your working directory.

The command displays the format for the target process if none of the following switches is used: /KILL, /ALL, /PREAMBLE, /SHARED, /UNSHARED, /DIRECTORY, or /FILENAME.

BREAKFILE Example

```
) BREAKFILE ↵
Error: No breakfile enabled for this ring
BREAKFILE
) BREAKFILE/SHARED 18 ↵

) BREAKFILE 18 ↵
Default Filename Specified
Ring:       7
Dump area specified:   Shared.
.
. (Processing and debugging occurs)...
.
) BREAKFILE/K 18 ↵
```

The response to the first command indicates that there is no break–file format yet defined for the current CLI process. The next BREAKFILE command requests the system to dump the shared area of the default ring to the default break file when PID 18 terminates abnormally. The next command verifies the current break–file setting.

The final command clears the current break–file format for PID 18.

BROADCAST *Macro*

Sends a message to all user processes on your system.

Format

BROADCAST *[message]*

This macro, supplied with the operating system, sends the text of message to all the processes on your system.

The CLI will replace any commas you include with a space. Do not include a semicolon or other character — like brackets or parentheses — that has special meaning to the CLI.

- No templates.
- No macro switches.
- Requirement: *Standard* (you must have Execute access to directory :UTIL and Read access to BROADCAST.CLI — these are true by default).
- See also: SEND, WHOS.

Why Use It?

Use the BROADCAST macro to send a message to all users. This macro is often used by the system operator to warn of impending shutdown.

Macro 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 and with this macro. This section also explains the CLI32 switch /STR=.

BROADCAST Example

```
) BROADCAST Shutdown for preventive maintenance in 10 minutes. Please log off.↵
.
.
(Display)
.
```

BROWSE *Utility*

Displays ASCII and binary files with search functions.

Format

XEQ BROWSE *[pathname] [...]*

The BROWSE utility lets you view or search through ASCII and binary files in a variety of display modes.

BROWSE opens as many as three files at one time, each in its own window. You can use template characters in the filename portion of the pathname argument (characters +, –, and *). If you use a template that matches more than one file, if you include more than one argument, or if you omit the pathname argument, BROWSE displays an initial screen allowing you to choose the first file to open.

By default, the initial cursor position is the end of the file and the cursor scrolls in a backward direction. (You can set initial position at the beginning and select forward display by executing BROWSE with the /FORWARD switch; from within BROWSE, you can change direction with the R command.)

A status line at the top of the screen indicates the current viewing direction, the filename, the file size, and the current position within the open file (the next word address. You can move forward and backward within the file using function or directional keys. You can set placemarks within files and move between placemark positions. You can search for a particular string, select a byte position, or move to the beginning, middle, or end of the file using commands and function keys.

The AOS/VS and AOS/VS II Menu–Based Utilities template names the function keys you can use with BROWSE; it is a very useful tool if you use BROWSE often. Even if you don't have a template, you can use the BROWSE functions by accessing its Command Menu (press function key 2, F2). You can obtain on–line help while in the BROWSE utility by using its H or ? command or the Help function key.

You receive BROWSE as a program file in :UTIL, filename BROWSE.PR. It runs on CRT terminals only, not on hardcopy terminals.

Why Use It?

BROWSE has many features that are not available in other CLI file display commands and utilities. With BROWSE, you can read, find, and copy strings within both data–sensitive and binary files, including files (such as system log or database files) that may already be opened by another program. You can open as many as three windows on one or more files, and switch windows with a keystroke. You can view a file backward (from end to beginning); this is useful for log files and for examining the last characters in very long files.

BROWSE Utility Switches

The BROWSE utility accepts the following switches. Many switch functions are duplicated in commands and key functions that you can use once you've started BROWSE. See Tables 5–2 through 5–7.

You can abbreviate all BROWSE switches to the shortest string that uniquely identifies them. In most cases, the shortest string is one character.

/B7 — Tells the program to ignore the parity bit (bit 8). The default is to display all 8 bits. You can change this from within BROWSE with the commands B8 and B7.

/ASKPM — If you include this switch, BROWSE will not automatically set placemarks; it will prompt for placemarks.

BROWSE lets you set up to 64 placemarks. By default (if you omit this switch) BROWSE sets one placemark (named a) at the end of each file read. Use the /FORWARD switch to set the default placemark at the beginning of the file. BROWSE names other placemarks as you set them, using digits 1, 2, 3, ... 9, lowercase letters b, c, ... z, and uppercase letters A, B, C, ..., Z, ., /, and ?. (Placemark naming is the only way in which BROWSE is case–sensitive.)

/COPY — Tells BROWSE to copy data exactly as it is in the source file, without adding a header or formatting the data. Normally (using the CO or nP commands), BROWSE writes information as it displays it, with header and screen control characters included. Use this switch when you want to copy data *exactly as is* to another file. The COPY function key performs the same function from within BROWSE.

/EBCDIC — Translates all data from EBCDIC to ASCII. Note that the translation remains in effect for *all* files in *all* windows, for the duration of this BROWSE session.

/FORWARD — Sets the initial position to the beginning of the file and the current direction to forward (opposite from the defaults). You can reverse direction from within BROWSE with the R command.

/LIST=pathname — Sets the pathname to be used when you copy information from a file. The default is the file pid.BROWSE.OUT (in the working directory), where pid is the PID number of the BROWSE process.

/MODE*[=n]* — If you specify *n*, sets the display mode to *n* (where *n* is a digit from 1 through 9. See the following "BROWSE Display Modes" section. Modes 1 and 2 are better for ASCII text. Modes 5 though 9 are better for binary files. You can set modes from within BROWSE with the Mn command. If you omit *n*, the utility will prompt for the display mode before displaying the file.

BROWSE Switches (continued)

/NC	No CLI access. Prevents access to the CLI from BROWSE. Use this when you run BROWSE from an application and want to deny users access to the CLI. Also, displays a tutorial–level status line that explains how to get help and how to exit from BROWSE.
/NF	No other file access. Allows the utility to open only the one file specified in the BROWSE command line. If the command line specifies more than one file, or if the program cannot open the file, BROWSE displays an error message and terminates. Use this when you run BROWSE from an application and want to restrict the user to just one file to which she/he has read access. The program also displays a tutorial–level status line that explains how to get help and how to exit.
/NP	No print or copy allowed. Tells BROWSE not to allow printing or copying during this session. Use this when you run BROWSE from an application and want to prevent the user from printing or copying text. The program also displays a tutorial–level status line that explains how to get help and how to exit.
/NU	New user. The program displays a tutorial–level status line that explains how to get help and how to exit from BROWSE.
/Q	Displays output in squeezed mode (with characters compressed) on terminals that support this feature.
/SHARED	Opens a file that another process already has open. This is useful for viewing current data in a file that another process (like a database manager) keeps open. The file size must be an even multiple of 2,048 bytes (it must be page aligned). Once browsing the shared file, you can choose the Wait option (command C7 or choice 7 on the Command Menu) to ask for notification that the file has grown. BROWSE will not otherwise tell you that the file has grown, although it will update the byte length number periodically.
/WSIZE=n	Set the initial window size to n lines. The default size is 23 lines. On a non–windowing terminal, BROWSE ignores this switch.
/ZEROS	Displays leading zeros in modes 5 through 9. By default, the program does not display them. You can control this from within BROWSE with the Z command.

BROWSE Display Modes

You can choose from nine different modes to display a file's information. Modes 1 through 4 display line–oriented text. Modes 5 through 9 display file address and numeric values of characters, similar to the DISPLAY program output. Mode 1 is the default. You can specify a non–default mode when you start BROWSE by using the switch /MODE=n. Once you've started the utility, you can select display modes in any of the following ways.

- Press the Display Mode function key (CTRL–F15), select a mode from the menu, then press NEW LINE.
- Enter C1, select a mode from the menu, and then press NEW LINE.
- Enter Mn, where n is the number of the mode you want.

The program redisplays the current screen, in the new mode, from the first line. The following examples of each mode use extracts from the CLI help file for the !USUBTRACT pseudomacro to illustrate the nine modes.

Display Mode 1 (Default Mode):

Displays file text line–by–line and converts CR and Form Feed characters to NEW LINE. Displays all 8 bits of each character (unless you set 7–bit mode with a switch or command, B7). Displays as periods (.) unprintable characters found in the file, including CTRL characters. (In this file, highlighting characters appear as periods.) If there are more than three blank lines in a row, displays the count instead of the blank lines. The status display (at top) indicates the total number of 16–bit words in the file (Size: 873), the next word (Next: 52), and the radix (Decimal).

```
<–Back–< CLI.PSM.USUBTRACT          Size: 873    Next: 52          Dec
```

*** Beginning of file ***

.!USUBTRACT Pseudomacro

Format:.
[!USUBTRACT integer1 integer2]
[!USUBTRACT integer integer [...]]. (CLI32 only)

Subtracts the value of one argument from another and displays the result. Each argument must evaluate to an unsigned decimal integer in the range 0 to 4,294,967,295. In CLI32, you can use more than two arguments.

If the difference between the two values is negative, the result is the absolute value of the difference modulo 4,294,967,296; that is, the remainder produced by subtracting the sum from 4,294,967,296.

.!USUBTRACT Pseudomacro Examples.

). WRITE [!USUBTRACT 17 5] <NL>.
12

). WRITE [!USUBTRACT 5000 199 25] <NL>.
4776

---------- End of Message ----------.

*** End of file ***

BROWSE (continued)

Display Mode 2:

Text appears as in Mode 1, except screen control characters (such as dim and blink CTRL sequences) are interpreted, not displayed as periods (.); they work the same way as with the TYPE command. Mode 2 does not count multiple blank lines; it displays all blank lines in the file. The total number of 16–bit words in the file (Size: 873), the next word (Next: 52), and the radix (Decimal), appear in the status display. If you use this mode with a binary file, results are unpredictable.

```
<–Back–< CLI.PSM.USUBTRACT          Size: 873    Next: 52          Dec
_____________________________________________________________________

*** Beginning of file ***
                          !USUBTRACT Pseudomacro
      Format:
      [!USUBTRACT  integer1  integer2]
      [!USUBTRACT  integer  integer  [...] ]              (CLI32 only)

   Subtracts the value of one argument from another and displays the result.
   Each ...
   ...
                      !USUBTRACT Pseudomacro Examples
      ) WRITE [!USUBTRACT 17 5] <NL>
      12
   ...
```

Display Mode 3:

Displays line–oriented text, but *all* unprintable characters, including NEW LINE delimiters, are not interpreted and appear as periods. There are exactly 80 characters per line. The total number of 16–bit words in the file (Size: 873), the next word (Next: 0), and the radix (Decimal), appear in the status display. Useful for reading the ASCII portions of symbol table files.

```
<–Back–< CLI.PSM.USUBTRACT          Size: 873    Next: 0           Dec
_____________________________________________________________________

*** Beginning of file ***
                        .!USUBTRACT Pseudomacro..   Format:..   [!USUBTRA
   CT integer1 integer2].   [!USUBTRACT integer integer [...] ].
      (CLI32 only)..   Subtracts the value of one argument from another and
   ...
    is, the remainder produced by subtracting the sum from 4,294,967,296...
              .!USUBTRACT Pseudomacro Examples...   ). WRITE
   [!USUBTRACT 17
   5] <NL>..   12..   ). WRITE [!USUBTRACT 5000 199 25] <NL>..   4776..

            ---------------------------- End of Message----------------------------.
*** End of file ***
```

BROWSE (continued)

Display Mode 4:

Displays text as mode 1 does, but standard delimiters appear as follows: NEW LINE = <nl>, CR = <cr>, form feed = <ff>, and tab = <tab>. Other unprintable (including CTRL) characters appear as <n>, where n is the octal ASCII value of the character, including parity. The total number of 16–bit words in the file (Size: 1551), the next word (Next: 102), and the radix (Octal), appear in the status line.

```
<–Back–< CLI.PSM.USUBTRACT          Size: 1551   Next: 102        Oct
--------------------------------------------------------------------------
*** Beginning of file ***
                    <35>!USUBTRACT Pseudomacro<34><nl>
   Format:<35><nl>
   [!USUBTRACT integer1 integer2]<nl>
   [!USUBTRACT integer integer [...] ]<34>          (CLI32 only)<nl>
<nl>
   Subtracts the value of one argument from another and displays the<nl>
...
   )<35> WRITE [!USUBTRACT 5000 199 25] <NL><34><nl>
   4776<nl>
<nl>
      ---------------------------- End of Message ---------------------------- <35>
```

Display Mode 5:

Displays octal file address and numeric values of characters with their ASCII values. The leftmost column indicates the word count for that line. The next eight columns are octal file address and numeric values of characters, paired in 16–bit words (each column is one–half word; there are four words per line). The rightmost column is the ASCII value (or periods, if unprintable characters). The total number of words in the file (Size: 664), next word (Next: 410), and radix (Oct), appear in the status line.

```
<–Back–< CLI.PSM.USUBTRACT          Size: 664    Next: 410        Oct
--------------------------------------------------------------------------
410 20155  67544  72554  67440  32054  31071  32054  34466 modulo 4,294,96
420 33454  31071  33073  20164  64141  72012  20040  20040 7,296; that.
430 20151  71454  20164  64145  20162  62555  60551  67144 is, the remaind
440 62562  20160  71157  62165  61545  62040  61171  20163 er produced by s
450 72542  72162  60543  72151  67147  20164  64145  20163 ubtracting the s
...
630 26455  26455  26455  26455  26455  20105  67144  20157 ----------End o
640 63040  46545  71563  60547  62440  26455  26455  26455 f Message ----------
650 26455  26455  26455  26455  26455  26455  26455  26455 ----------
660 26455  26455  26455  26455  35                          ----------.

*** End of file ***
```

BROWSE (continued)

Display Mode 6:

Displays decimal file address and numeric values of characters with their ASCII values. The leftmost column indicates the word count for that line. The next eight columns are file address and numeric values of characters, paired in 16–bit words. The rightmost column is the ASCII value (or periods, if unprintable characters). The status line shows the total words (Size: 664), next word (Next: 264), and radix (Dec).

```
<–Back–< CLI.PSM.USUBTRACT          Size: 436    Next: 264       Dec
-------------------------------------------------------------------

264 8301   28516 30060 28448 13356 12857 13356 14646 modulo 4,294,96
272 14124  12857 13883 8308  26721 29706 8224  8224  7,296; that.
280 8297   29484 8308  26725 8306  25965 24937 28260 is, the remaind
288 25970  8304  29295 25717 25445 25632 25209 8307  er produced by s
296 30050  29810 24931 29801 28263 8308  26725 8307  ubtracting the s
...
408 11565  11565 11565 11565 11565 8261  28260 8303  ----------End o
416 26144  19813 29555 24935 25888 11565 11565 11565 f Message --------
424 11565  11565 11565 11565 11565 11565 11565 11565 ----------
432 11565  11565 11565 11565 29                      ----------.

*** End of file ***
```

Display Mode 7:

Displays hexadecimal file address and numeric values of characters with their ASCII values. The leftmost column indicates the word count for that line. The next eight columns are file address and numeric values of characters, paired in 16–bit words. The rightmost column is the ASCII value (or periods, if unprintable characters). The status line shows the total words (Size: 1B4), next word (Next: 108), and radix (Hex).

```
<–Back–< CLI.PSM.USUBTRACT          Size: 1B4    Next: 108       Hex
-------------------------------------------------------------------

108 206D   6F64  756C  6F20  342C  3239  342C  3936  modulo 4,294,96
110 372C   3239  363B  2074  6861  740A  2020  2020  7,296; that.
118 2069   732C  2074  6865  2072  656D  6169  6E64  is, the remaind
120 6572   2070  726F  6475  6365  6420  6279  2073  er produced by s
128 7562   7472  6163  7469  6E67  2074  6865  2073  ubtracting the s

198 2D2D   2D2D  2D2D  2D2D  2D2D  2045  6E64  206F  ---------- End o
1A0 6620   4D65  7373  6167  6520  2D2D  2D2D  2D2D  f Message --------
1A8 2D2D   2D2D  2D2D  2D2D  2D2D  2D2D  2D2D  2D2D  ----------
1B0 2D2D   2D2D  2D2D  2D2D  1D                      ---------.

*** End of file ***
```

BROWSE (continued)

Display Mode 8:

Similar to mode 5, but display shows bytes (16 per line) instead of 2–byte words. There is no room for an ASCII field on the right. The status line shows the total words (Size: 664), next word (Next: 400), and radix (Oct).

```
<–Back–< CLI.PSM.USUBTRACT          Size: 664   Next: 400       Oct
-------------------------------------------------------------------

400  146 40  164 150 145 40  144 151 146 146 145 162 145 156 143 145
410  40  155 157 144 165 154 157 40  64  54  62  71  64  54  71  66
420  67  54  62  71  66  73  40  164 150 141 164 12  40  40  40  40
430  40  151 163 54  40  164 150 145 40  162 145 155 141 151 156 144
440  145 162 40  160 162 157 144 165 143 145 144 40  142 171 40  163
...
640  146 40  115 145 163 163 141 147 145 40  55  55  55  55  55  55
650  55  55  55  55  55  55  55  55  55  55  55  55  55  55  55  55
660  55  55  55  55  55  55  55  55  35

***End of file***
```

Display Mode 9:

Similar to mode 6, but display shows bytes (16 per line) instead of 2–byte words. There is no room for an ASCII field on the right. The status line shows the total words (Size: 436), next word (Next: 264), and radix (Dec).

```
<–Back–< CLI.PSM.USUBTRACT          Size: 436   Next: 264       Dec
-------------------------------------------------------------------

264  32  109 111 100 117 108 111 32  52  44  50  57  52  44  57  54
272  55  44  50  57  54  59  32  116 104 97  116 10  32  32  32  32
280  32  105 115 44  32  116 104 101 32  114 101 109 97  105 110 100
288  101 114 32  112 114 111 100 117 99  101 100 32  98  121 32  115
296  117 98  116 114 97  99  116 105 110 103 32  116 104 101 32  115
...
416  102 32  77  101 115 115 97  103 101 32  45  45  45  45  45  45
424  45  45  45  45  45  45  45  45  45  45  45  45  45  45  45  45
432  45  45  45  45  45  45  45  45  29

***End of file***
```

BROWSE Commands

Command Menu

1 – Display Mode
2 – Placemarks (view or change)
3 – Go To (change file position)
4 – Find a string
5 – Find a string and count hits
6 – Read a file
7 – Wait for file size to change
8 – Execute a CLI
9 – Open Files in Windows
0 – Global Find
C – Column Highlight
L – List (File List Maintenance)
O – Output (Print or Copy) to a file
U – Get file UDA

Choice: ?

At the *Choice: ?* prompt, enter any command character, or type H or ? for help. You can execute any of these commands directly by entering the command Cn, where n is a command menu item (1–9, 0, C, L, O, or U).

In addition to the commands shown on the Command Menu, there are other function key, CTRL, or direct keystroke commands that you can enter at any point after starting the BROWSE utility. Tables 5–2 through 5–7 describe the commands and function keys you can use for changing cursor position within a file, searching and marking text, altering the display, moving between open files, copying or printing file contents, exiting from the BROWSE program, or getting help.

Table 5–2 Leaving BROWSE or Getting Help

You Type This	What It Does
BREAK/ESC or Cancel/Exit (F11)	Exits from current menu, or if BROWSE is not displaying a menu, terminates the program.
CLI (SHIFT–CTRL–F5)	Starts a new CLI process on your terminal. To terminate the CLI and return to BROWSE, enter BYE.
Command (F2)	Displays the Command Menu.
CTRL–C CTRL–A	Terminates the program.
CTRL–D	Terminates the program.
H	Displays the Help menu.
Help (SHIFT–F1)	Displays on–line Help. Use Cancel/Exit to exit.
Terminate (CTRL–SHIFT–F11)	Exits the program from anywhere.

Table 5–3 Changing Position Within a BROWSE File

You Type	What It Does
. (period) or >	Moves the cursor (scrolls) one half screen forward.
, (comma) or <	Moves the cursor (scrolls) one half screen backward.
↓ (Downarrow)	Moves forward to the next line of information.
↑ (Uparrow)	Moves backward to the previous line of information.
⤵ (New Line)	Moves to the next line in the current direction: forward if direction is *Fore*; backwards if direction is *Back*. You can reverse direction with the R command. The current direction appears on the status line at the top of the screen.
Back Field (SHIFT–F11)	Moves cursor to the previous input field at menu prompts.
Execute (F1)	Displays the next or previous screen of information, depending on the current direction. See also Column Highlights, Table 5–6.
Gx	Goes to a file position: GB to the beginning, GM to the middle, GE to the end, GF to form feed (next page), GU to the file UDA.
Go To (SHIFT–F5)	Lets you change position in the file by selecting the new location. Same as C3, choice 3 on the Command Menu.
nL	Moves forward (in the current direction) n lines (+nL), or backward (opposite the current direction) n lines (–nL). You can use k to specify multiples of 1,024; for example, –16k means go back 16,384 bytes. BROWSE defines a line as one line displayed on the screen (including one that says *line repeats n times*), not as the number of lines in the file. Therefore, nL commands may not be behave precisely as you expect.
Next Page (CTRL–F4)	Displays the next page boundary (form feed) plus the text lines before and after the boundary.
Next Screen (F4)	Displays the next screen of information.
Px	Moves your position in the file to the placemark named by x.
Placemark (CTRL–F13)	Lets you go to a placemark (position in the file) using the Placemark screen. The default placemark is named a and is set at the end of the first file you open (or the beginning if you used the /FORWARD switch). See Placemark and Instant Placemark in Table 5–4.
R	Reverses the current direction, and presents one line of data in the new direction.

Table 5–4 Marking and Searching Text with BROWSE

You Type This	What It Does
CTRL–F	Global find. If a find string has been defined, skips the remainder of this file and searches the next file in the list. Sets a placemark at the first occurrence of a string it finds in any file other than the current one. If no find string is defined, same as F; see F.
F (Find)	Displays a menu, then searches for a specified character string. After the program finds a string once, enter F to continue the search for it. If there is a file list, use F to search the other files. (You can create a file list with the CL command, the L command, or from the Command Menu. See Table 5–5.) The program does not find a string if it is broken by a line delimiter; for example, if you specify cash□ flow, it would find cash □flow but not cash↵ flow. To search for a number, specify a Binary search. You can search for one or more characters (including nonprinting characters), by entering each character in the form <n>, where n is ASCII value of the character, and then specifying a Binary search. To search for the number of *occurrences* of a string, use the command C5 or item 5 from the Command Menu. Instead of the F command, you can use the Find function key (SHIFT–F6) or enter C4 or / (slash). To change the search string, use the Find function key.
Instant Placemark (CTRL–SHIFT–F13)	Sets a new placemark at this screen without interaction. See also Placemark.
Placemark (CTRL–F13)	Lets you set, delete, create, or go to a placemark (position in the file) at the Placemark screen. The default placemark for the first file opened is named a and is set at the end of the file — or the beginning if you used the /FORWARD switch. BROWSE also sets a placemark at the first occurrence of a string it finds in any file other than the current one. BROWSE names each additional placemark as follows: 1, 2, 3, 4, 5, 6, 7, 8, 9, b, c, ... z, A, B, C, ... Z. To go to a placemark, use the Px command (x is the placemark name), the GO TO function, or Placemark function key. To set a placemark, use the Placemark screen or Instant Placemark.

Table 5–5 Working With BROWSE Windows and File Lists

You Type This	What It Does
Delete (F7)	From the File List Maintenance screen (CL, or item L from the Command Menu), deletes one or more names from the file list for global searches (see L, File List Maintenance). During column highlighting removes a column highlight (see Column Highlight, Table 5–6).
Index (SHIFT–F2)	From the File List Maintenance screen (Open Files in Windows (F15)), displays the file list. See also L.
Insert (F6)	Adds a filename to the file list (see L command or Read) or for column highlighting (see Column Highlight, Table 5–6).
L (File List Maintenance)	BROWSE uses the file list for global searches. From the Command Menu (or CL), this command lets you add and remove names from the file list.
Next File (F10)	Displays the next file in the file list. On an empty list, displays the Open Files in Windows screen so you can create a list.
Open Files in Windows (F15)	Lets you assign or change file/window specifications by prompting for filenames, window sizes, and margins.
Previous File (F9)	Displays the next filename in the file list. On an empty list, displays the Open Files in Windows screen so you can create a list.
Read (F12)	Lets you change the file in the current window. The program prompts for filename, window size, margin, and compressed mode selection. You can open files in up to 3 windows. After you answer the filename prompts, the program opens the file in a window. You can switch betweem open windows with the Switch Windows function key.
Switch Windows (CTRL–F1)	Displays the next window. If only one window is open, the program prompts you for filenames, window sizes, and margins so it can open a file in another window. If more than one window is open, the number of the current window appears in the left corner of the status line.

Table 5–6 Altering the BROWSE Display

Type This	What It Does
B7, B8	Tells the program to ignore the parity bit (B7) or to display it (B8). The default is to display the parity bit.
Column Highlight (CTRL–F2)	Lets you highlight one or more column positions (useful for checking column alignment) or remove highlighting. To add column highlighting, press the Column Highlight key, then specify each column you want to highlight: press the right– or leftarrow key to set position and then press S (set) or the Insert function key. After specifying all columns to highlight, press Execute (F1). To remove highlighting from a column, press the Column Highlight key, then set position on the column and press the Delete function key. To remove all highlights, press the Column Highlight key and then the Erase key.
CTRL–R	Refreshes the current window. In shared mode, tells the program to reread the file and update the screen.
Display Mode (CTRL–F15)	Lets you change display mode at the Display mode menu. Same as Mn, where n is the number of the mode you want.
Erase (SHIFT–F14)	During column highlighting, removes all highlighting; see Column Highlight.
ERASE PAGE or CTRL–L	Refreshes screen. Useful if your screen becomes garbled (over a modem, for example). Also, centers the window if your terminal does not support windows and you have the window marker on your screen.
Interrupt (F5)	Resets continuous scrolling (started by Execute key).
Mn	Changes display mode to n (0 – 9); see Display Modes.
Open Files in Windows (F15)	Lets you assign or change file/window specifications by prompting for filenames, window sizes, and margins.
S	Sets a column highlight; see Column Highlight, above.
W (Wait)	Tells the program to wait until the file grows. It displays an appropriate screen, which you can interrupt at any time. This state is useful when you are browsing a log file.
Z (Zero display)	Turns leading zero suppression on or off (toggles). This is meaningful in modes 5 through 9 only.

Table 5–7 Copying and Printing Text from BROWSE Files

Type This	What It Does
Copy (CTRL–F8)	Changes (toggles) the output mode between print and copy for copying screen text onto a file. BROWSE shows the output mode with a C or P at the left corner of the status line. In Print mode, BROWSE writes the screen as you see it, including the header line and screen control characters (such as dim). In Copy mode, the program copies data raw from the source file, without formatting or header line. To write the text to the file, use the O command from the Command Menu, Save function key, CTRL–P, or the nP command, where n is the number of lines you want written to the file. The filename is pid.BROWSE.OUT (where pid is the PID). If the file already exists, BROWSE appends to it. To specify a filename, use the CO command.
CO (Output)	Lets you specify a file for output. BROWSE asks for a filename, and then asks if you want Print or Copy. Copy writes information as it is in the file; Print copies it as displayed on the screen, with the BROWSE status line and screen control characters, if any.
CTRL–P	Same as Save (F13); see Save.
nP	Prints or copies n lines to a file, starting with the line shown at top of the screen. With print, BROWSE writes lines as shown on the screen; with copy, it writes them as they are in the file. You can select print or copy with the COPY function key; it displays P or C at the left of the status line. BROWSE creates the file with the filename n.BROWSE.OUT, where n is the PID of the BROWSE process.
Save (F13) or CTRL–P	Prints or copies lines to a file, starting with the line shown at top of the screen. In print mode, BROWSE writes lines as shown on the screen; in copy mode it writes them as they are in the file. You can set print or copy mode with the COPY function key, as shown with P or C at the left of the status line. BROWSE creates the file with filename is n.BROWSE.OUT, where n is the BROWSE PID. If you want to specify a different filename, use the CO command.

BROWSE Example

The following example shows a BROWSE session (in default display mode) that includes access to the Command Menu, changing position with Go To, finding a string, setting an instant placemark, and exiting to the CLI.

) BROWSE MARK_II ↵	(Execute the utility without switches on the file MARK_II)
<–Back–<MARK II< Size: 107557 Next: 107557 Dec	(Program displays status line and first screen of text. The default position is at the end of the file.)
...	
End of Mark_II Outline	
Command (F2 function key)	(Press the Command function key.)
Command Menu *1 – Display Mode* *2 – Placemarks (view or change)* *3 – Go To (change file position)* *4 – Find a string* .... *Choice:* 3 ↵	(Program displays Command menu.) (Select choice 3, Go To.)
Go to byte: *Enter 'Help' for help* *Default radix is decimal.*	(Program prompts for position.)
Go to byte: B ↵	(Enter B for beginning of file.)
MARK II >–Fore–> Size: 107557 Next: 1104 Dec	(Status line)
...	(First window of text in file.)
F	(Enter F to find string.)
Find: Precondition ↵	(Program prompts for, and user enters, sought string.)
Case Sensitive: N	(Enter N for case–insensitive search.)
Binary (numeric)? No ↵	(Not a binary search; select default, No.)
MARK II >–Fore–> Size: 107557 Next: 15414 Dec	(Status line; string found.
... precondition...	(Program displays all occurrences of the
...	string in a window.)
Preconditions...	
Instant Placemark (CTRL–SHIFT–F13)	(Set an instant placemark, number 1, at the first occurrence of the sought string; you can return later with command P1.)
...	(Continue browsing file.)
Cancel/Exit (F11)	(Exit to the CLI using Cancel/Exit key.)
)	

BYE *Command*

Terminates this CLI process.

Format

BYE *[argument] [...]*

This command ends the current CLI session. The CLI process terminates. If EXEC is the father process of the CLI, this command logs you off the system, otherwise you return to the process that created the CLI.

If you supply one or more arguments, the CLI passes the argument string to its father process.

If the CLI process any son processes when you enter this command, the following message appears:

You have sons. Do you want to terminate?

Respond YES (to terminate the CLI and its son processes) or NO. With CLI32, you can suppress this message by including the /TERMINATE switch.

If the CLI process is the master CLI (PID 2), BYE starts the system shutdown sequence. Generally, before typing BYE to PID 2, you should use the DOWN macro. For more information about system shutdown, see the "Installing" manual for your operating system.

After you type BYE, CLI process termination occurs as follows:

1. The CLI executes your LOGOFFMACRO, if any
2. The CLI terminates any son processes (asking for confirmation first, shown above)
3. The CLI terminates.

- No templates.
- No argument switches.
- Requirement: *Standard.*
- See also: CHAIN, LOGOFFMACRO, PROCESS, TERMINATE, XEQ.

Why Use It?

Use the BYE command when you have finished using the CLI and want to terminate the CLI process. Using the this command is the preferred method for ending a CLI session.

The LOGOFFMACRO command in CLI32 provides a convenient way to specify a macro that the CLI will invoke in response to your BYE command.

BYE 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=.

You can use only one of the following switches at a time.

/ABORT	Displays an abort message at termination.
/ERROR	Displays an error message at termination.
/WARNING	Displays a warning message at termination.
/TERMINATE	(CLI32 only.) If the process has sons, does not issue the *You have sons* confirmation message; the process terminates without interaction.

BYE Example 1

```
) BYE ↵
AOS/VS CLI  Terminating  12-DEC-90  15:14:18
```

This command ends the current CLI session.

BYE Example 2

```
) BYE/WARNING All done! ↵
AOS/VS II CLI  Terminating  3-JAN-90  16:22:34

*Warning*
All done!
Warning: From Program
```

This BYE command returns to a father CLI process.

CHAIN *Command*

Replaces the current CLI process with a specified program.

Format

CHAIN program–pathname *[argument] [...]*

This command executes the specified program, replacing the current CLI with the process run from program–pathname. You can include any arguments that you want passed to the new process.

If the process you chain from is your initial CLI process, when the chained process terminates you will be logged off.

You can use any arguments or switches appropriate for the program named in program–pathname. The program you are executing can access the arguments and switches with the ?GTMES system call. (For more information about the ?GTMES call, see *AOS/VS, AOS/VS II, and AOS/RT32 System Call Dictionary, ?A through ?Q.*)

- No templates.
- Use any argument switches appropriate to the new program.
- Requirement: *Standard.*
- See also: EXECUTE/XEQ and PROCESS (to run a program as a subordinate process without replacing the current CLI).

Why Use It?

You can use the CHAIN command to execute another program without returning to the CLI. You will find this command useful if, for example, your system is approaching its maximum number of processes, or if you have already created your allotted number of son processes.

Chaining to another program can release the system resources that were being used by your CLI process.

CHAIN 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=.

/D	Starts the new process in the assembly language debugger.
/DEBUG	(CLI32 only). Same as /D.

CHAIN Example

) CHAIN MYPROG ↵

This program replaces the current CLI process by overwriting it with MYPROG.PR. When you exit MYPROG, you either return to the process that executed the CLI, or if the CLI were a son of the EXEC process, you are logged off the system.

CHARACTERISTICS *Command*

Displays or sets the characteristics for a character device.

Format

CHARACTERISTICS *[device] [...]*

The characteristics of a device determine how it interprets input and displays output. This command displays or sets characteristics for your terminal (if you omit the device arguments) or for the specified character devices. Without command switches, the CLI displays the current characterics of the specified device. You must have System Manager privilege turned on or use the PID 2 process to view the characteristics of a console owned by another user's PID.

To set characteristics, use command switches. Many switches represent an attribute that you can turn on or off. To turn attributes off, specify the /OFF switch followed by one or more switches for attributes that you want to turn off. You can use /ON in the same way to turn attributes on. If you omit /OFF or /ON, the default value is ON. So to turn one or more characteristics on, you can omit the /ON switch For example:

CHARACTERISTICS/xxx/yyy (turns characteristics xxx and yyy on)
or
CHARACTERISTICS/xxx/yyy/OFF/zzz (turns characteristics xxx and yyy on and zzz off)

You need specify only the values that you want to change. If you do not include the switch for a certain setting, the value of that setting in the current CLI environment is used.

NOTE: If you set characteristics, they are effective until you change them, log off the system, or move (via the POP or PUSH commands) to an environment that has different characteristics.

- No templates.
- No argument switches.
- Requirement: *Standard,* except for changing a device's default characteristics with the /DEFAULT switch (which requires *PID 2*), or viewing the characteristics of a device owned by another user's pid (which requires *PID 2* or *System Manager* privilege).

Why Use It?

Use the CHARACTERISTICS command to control how the system sends data to or receives data from a device. For example, you may want to set page mode for your terminal so that it displays one page of data at a time and waits for you to press CTRL–Q before it displays subsequent data.

If your terminal is connected to your system via a modem, you can use this command to set your terminal's characteristics so that it can send and receive data appropriately over the modem connection.
If normally you do not want to use the default characteristics assigned for your terminal, you can include a CHARACTERISTICS command in your startup macro to reset them each time you log on.

From PID 2, before EXEC has enabled terminals, you can use the CHARACTERISTICS/DEFAULT command to change the default characteristics for any terminal. It is most convenient to do this in the UP.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 switch /STR=.

Remember that you can turn on or off one or more switches using the form

CHARACTERISTICS/switches (or /CHARACTERISTICS/ON/switches) (to turn on)
or
CHARACTERISTICS/OFF/switches (to turn off)

/8BT	ON:	Interprets all 8 bits of an ASCII character as data. Use this with 8–bit character sets. The following octal codes will echo as an uparrow followed by an alphabetic character (they will echo as CTRL characters) regardless of this switch: 1 to 10; 13; 16 to 32; 34 to 37. On a VT100–compatible terminal, select this for International VT100 support; XLT must also be selected. VT100 support works with AOS/VS II on a newer asynchronous controller only.
	OFF:	Interprets the last 7 bits as data (default).
/16B	ON:	Enables Asian language translation. Asian language support must have been selected at system generation. Setting this characteristic enables recognition of related switches like /G1G0 and /NLX.
	OFF:	Does not enable Asian character translation.

CHARACTERISTICS (continued)

/4010I — Identifies the device as a DGC Model 4010I (CRT1) terminal.

/6012 — Identifies the device as a DGC Model 6012 (CRT2) terminal.

/605X — Identifies the device as a DGC DASHER Model 6052 or 6053 (CRT3) terminal; also applies to DASHER D210 and D211 terminals.

/6130 — Identifies the device as a DGC DASHER Model 6130 (CRT6) terminal; also applies to DASHER D410 and D460 terminals.

/ACC — Tells the system that the line requires modem access control. A user needs a modem privilege to log on over this line. (Also see /MOD.)

/AUTOBAUD — Tells the system to automatically determine your terminal's baud rate. The system detects the following baud rates:

300	1200	2400	9600
600	1800	4800	19200

If you select /AUTOBAUD, type three NEW LINE characters before logging on so that the system can determine and set the correct baud rate. (Use the Carriage Return key if your terminal doesn't have a NEW LINE key.)

If an application (like DG/BLAST or DG/GATE) tries to log on, it must not send NEW LINE characters faster than one per .1 second.

If this attempt to determine the baud rate and log on doesn't seem to work, do the following:

1. Be sure that /AUTOBAUD is enabled for the line, either through CHARACTERISTICS/ON/AUTOBAUD or at system generation time.
2. Be sure that the terminal is set to one of the supported baud rates and you or a remote machine's software isn't entering NEW LINE or Carriage Return characters too quickly.
3. Press the NEW LINE or Carriage Return key three more times.
4. Press CMD BREAK/ESC or CTRL–C CTRL–B to send a BREAK character sequence; then try step 3 again.
5. If step 4 fails after two or three attempts, your system's operator may need to disable and then enable the console line.

The default value of /AUTOBAUD is off.

CHARACTERISTICS (continued)

/BAUD=n

Sets a device's baud rate (bits per second) to one of the following values (IAC, LAC, and USAM systems only).

45.5	300	3600
50	600	4800
75	1200	7200
110	1800	9600
134.5	2000	19200
150	2400	38400

/BREAK=value

Specifies the way the system responds to the break sequence. This involves the CMD and BREAK/ESC keys or the BREAK key. The following values are valid.

BMOB (The default.) Clears binary mode and restores normal control character handling.

CAOB Issues a CTRL–C CTRL–A console interrupt sequence (^C^A is not echoed on the terminal).

CBOB Issues a CTRL–C CTRL–B console interrupt, aborting the process (^C^B is not echoed on the terminal).

CFOB Issues a CTRL–C CTRL–F console interrupt sequence (enabling your application program to detect a break).

DCOB Disconnects the user process and logs the user off; also, disconnects a modem.

/CALLOUT

ON: Enables host–initiated calls, if the controller allows this).

OFF: Does not enable host–initiated calls (default).

In other words, use /CALLOUT to tell the system to initiate calls and /OFF/CALLOUT to tell the system to prevent them.

/CHARLEN=n

Sets the character length in bits, including stop bits. n can be 5, 6, 7, or the default value of 8. (IAC and USAM systems only.)

CHARACTERISTICS (continued)

CONTYPE=value

On AOS/VS II only, the value tells what kind of terminal connection this is. Values are set by the system and you cannot change them. Values and their meanings are

BITMAPPED (windowing terminal).

DIRECT (standard connection to asynchronous controller).

PAD (connection via Packet Assembler/Disassembler hardware).

PBX (connection via Private Branch Exchange controller — a PIM or CPI/24 controller).

PCVT (connection via DG/PC*I personal computer integration controller).

TERMSERVER (connection to Termserver hardware).

TELNET terminals (connection via TCP/IP TELNET via Intelligent LAN controller, ILC; if connected via TCP/IP TELNET and Termserver hardware, these appear as type TERMSERVER).

VIRTUAL (connection via virtual terminal agent).

/CPL=n

Specifies the maximum number of characters per line. The default value is n = 80 and the range is 8 through 255. The system wraps or truncates lines longer than n, depending on the value of /EOL. See also /WRP.

/CRTn

Specifies a nonstandard CRT type, where n is 4, 5, or a number from 7 through 15.

/CTD

Tells the system that this is a contended line. When you log on, the system will disconnect the line if you don't respond within a given period.

/DEFAULT

If used alone, displays the default characteristics of your terminal. From PID 2 (or with System Manager Privilege on), you can set the default characteristics for other devices. You must do so before EXEC enables the device. An example is

```
) CHAR/DEFAULT/MOD/MRI/BAUD=1200 @CON14 ↵
```

The new default characteristics take effect the next time the EXEC program enables the line. To change the default characteristics of a device after it has been enabled, disable the device, wait for a *Console disabled* message, issue the command, and enable the device.

CHARACTERISTICS (continued)

/DKHW

(CLI32 and AOS/VS II only.) Relates to Kanji character support.

ON Disables half–wide character support (use this for Chinese, Korean, and Taiwanese character sets).

OFF Enables half–wide character support.

The system can support half–wide characters only if Asian language support was selected for the terminal controller during system generation and if /DKHW is OFF. You must also use /16B and /8BT.

/EB0, /EB1

Specify the echoing of control characters and the ESC key according to the following switch settings:

/EB0	/EB1	Meaning
ON	ON	Reserved.
ON	OFF	Echoes control characters as ^x (x is the character); echoes ESC as $. (This is the default.)
OFF	ON	Echoes characters exactly as entered, without interpretation.
OFF	OFF	Does not echo any characters.

/EOL

ON: Does not output a NEW LINE if the number of characters exceeds the current line length. In other words, truncate lines if they have more than /CPL characters. See also /WRP.

OFF: Outputs a NEW LINE if the number of characters exceeds the current line length; this wraps the line after /CPL characters. By default, /EOL is off. See also /WRP.

/EPI

Has no effect; formerly used with the AOS operating system.

/ESC

ON: Interprets the ESC or BREAK/ESC key as a console interrupt (CTRL–C CTRL–A) sequence.

OFF: Does not interpret the ESC or BREAK/ESC key as a console interrupt sequence. This is the default value.

/FF

ON: Outputs a form feed when the device is opened.

OFF: Does not output a form feed when the device is opened (default).

CHARACTERISTICS (continued)

/FKT

ON: Allows function keys to serve as delimiters in data–sensitive read operations.

NOTE: The CLI will not operate correctly with /FKT on.

OFF: Does not interpret function keys as delimiters in data–sensitive read operations.

/G1G0

(CLI32 and AOS/VS II only.) Relates to Taiwanese character support.

ON Enables the G1G0 character set (for Taiwanese characters).

OFF Disables the G1G0 character set.

The system can support the G1G0 character set only if if Asian language support was selected for the terminal controller during system generation and if /G1G0 is ON. You must also use /16B and /8BT.

/HARDCOPY

Identifies the device as a hardcopy (printing) terminal.

/HDPX

ON: Tells the system to provide half–duplex support on a modem–controlled line. Also, the line must support RS–232 modem control signals and you must set /SMCD.

OFF: The default mode; this has the system provide full–duplex support on a modem–controlled line.

/HIFC

ON: Enables hardware input flow control by telling the system to use the RTS (Request to Send) signal to stop input from the device attached to your terminal line. Specify this value if the device supports RTS/CTS (CTS = Clear to Send) flow control. If you set /HIFC you cannot also set /HDPX or /MOD. This characteristic is supported on certain asynchronous controller lines only.

OFF: Disables hardware input flow control. This is the default value.

/HOFC

ON: Enables hardware output flow control by telling the system that the device attached to your terminal line uses the RTS (Request to Send) signal to stop system output. Specify this value if the device supports RTS/CTS (CTS = Clear to Send) flow control to stop system output (some modems or printers). This characteristic is supported on certain asynchronous controller lines only.

OFF: Disables hardware output flow control (default).

CHARACTERISTICS (continued)

/IFC

ON: Enables software input flow control by telling the system to use CTRL–S and CTRL–Q (X–ON/X–OFF) to control input from this line. Specify this for an asynchronous line connected to a device that supports software flow control. This characteristic, with /OFC, is most useful for communications between systems.

OFF: Disables software input flow control (default).

/KVT

ON: Enables support for Kanji VT100 terminals when used in conjunction with /XLT

OFF: Disables support for Kanji VT100 terminals.

/LEVEL=n

(CLI32 only.) Sets the characteristics to those established at level n. No other switches are allowed.

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.

/LPP=n

Specifies the number of lines per page (4 through 255); the default for display terminals is 24 lines. Don't use the /OFF switch with /LPP.

/MDUA

ON: Allows you direct access to the modem on this line. Your program can issue ?WRITE system calls to send data over the line before a connection is established on the line. /MDUA allows a program such as DG/BLAST, for example, to send commands to a modem. /MDUA has no effect unless /MOD is also set.

OFF: Does not allow you direct access to the modem on this line. This is the default value.

/MOD

ON: Specifies the use of a modem interface on this line. Use /MOD for only for a line that supports modem signals. For an auto–answer modem, you must also set /MRI. Generally, you will also want to set /HOFC and you may want to set the baud rate (/BAUD=). An example of a command is

```
) CHARACTERISTICS/DEFAULT/MOD/MRI&
/BAUD=1200 @CON4 )
```

/MOD includes the functionality of /ACC and /CTD.

OFF: Specifies no modem interface on this line. This is the default value.

CHARACTERISTICS (continued)

/MRI

ON: Tells the system to monitor the ring indicator (from the modem) and to assert certain signals if the telephone rings. This monitoring and asserting occurs only when the line is connected to a modem and /MOD is on. Generally, you can turn /MRI on for any modem–connected line; it is essential that you do this on CCITT–standard lines often used in Europe.

OFF: Does not monitor the ring indicator or does not assert any signals if the telephone rings. This is the default value.

/NAS

ON: Specifies the device as non–ANSI standard. Use this with terminals that have Carriage Return and line feed keys. On input, a Carriage Return is converted to a NEW LINE, and a line feed is converted to a Carriage Return. On output, a line feed is converted to a Carriage Return followed by a line feed.

OFF: Specifies the device as ANSI standard. This is the default value.

/NLX

(CLI32 and AOS/VS II only.) Relates to Asian natural language translation; applies primarily to PC connections where the PC provides natural Asian language translation.

ON: Disables natural Asian language translation.

OFF: Enables natural natural language translation. Support for Asian language translation must have been selected at system generation. You must also use /16B and /8BT.

/NNL

This switch applies only to AOS/VS since AOS/VS II does not support card readers; it works in CLI16 only.

ON: Does not automatically append NEW LINES to card images.

OFF: Appends a NEW LINE to each card image. This is the default value.

/NRM

ON: Suppresses messages sent by users other than PID 2.

OFF: Allows reception of messages from all users. This is the default value.

CHARACTERISTICS (continued)

/OFC

ON: Enables software output flow control by telling the system to use CTRL–S and CTRL–Q (X–ON/X–OFF) to control output to this terminal. Specify this when you run a program that uses binary I/O and you do not expect the data to contain CTRL–S and CTRL–Q.

OFF: Disables software output flow control. This is the default value.

/OFF

Turns off the characteristics indicated by the switches that follow (up to /ON or a delimiter). An example is

```
) CHARACTERISTICS/OFF/NRM/MOD ↵
```

to turn off characteristics /NRM and /MOD regardless of their prior states.

/ON

Turns on the characteristics indicated by the switches that follow (up to /OFF or a delimiter). If you omit /OFF, the system assumes /ON. /ON is most useful after you've used /OFF in a command. An example is

```
) CHARACTERISTICS/OFF/NRM/ON/PM ↵
```

to turn off characteristic /NRM regardless of its prior state and to turn on characteristic /PM regardless of its prior state.

/OTT

ON: Converts octal 175 ({) and 176 (~) output to octal 33 (ESC, which echoes as $). You may want to use this with a VT100–compatible terminal; see Appendix D.

OFF: Does not convert an octal 175 or 176 value on output.

/P*[=n]*

Without =*n*, sets the characteristics to those used in the previous environment level. With =*n* (CLI32 only), sets the characteristics to those used in the specified environment level. The *n* specifies the number of levels above the current level (toward 0). No other switches are allowed. Same as /PREVIOUS*[=n]*.

/PARITY=p

Sets the parity rate to the specified value of p, where p is ODD, EVEN, or NONE (hardware–controlled systems only). The default value is NONE. Do not use the /OFF switch with /PARITY.

/PBN

This switch applies only to AOS/VS since AOS/VS II does not support card readers; it works in CLI16 only.

ON: Uses packed format on binary read (which places 4 card columns into 3 words).

OFF: Right justifies card columns in memory.

CHARACTERISTICS (continued)

/PM

ON: Enables page mode, which suspends output after n lines are displayed; CTRL–Q resumes output. n is the value of /LPP= or the next FORM FEED character, whichever comes first. The default value of n is 24, but you can change it with a command of the form

CHARACTERISTICS/LPP=n

OFF: Disables page mode, which allows continuous output. This is the default value.

/PREVIOUS*[=n]*

(CLI32 only.) Same as /P*[=n]*. No other switches are allowed. ■

/RAC

ON: Sends 2 delete–character codes after each NEW LINE and Carriage Return character (for slow hardcopy terminals).

OFF: Does not send any delete–character codes after each NEW LINE and Carriage Return character (default).

/RAF

ON: Sends 17 DEL (delete) characters after each form feed character (for slow hardcopy terminals). The system ignores /RAF if /SFF is on.

OFF: Does not send DEL (delete) characters after a form feed character (default).

/RAT

ON: Sends 2 delete–character codes after each TAB=CTRL–I character (for slow hardcopy terminals). The system ignores /RAT if /ST is on.

OFF: Does not send any delete–character codes after a tab (default)

/RESET

Sets the characteristics of the device to its default settings. You must use this switch alone. To display the default settings, use /DEFAULT alone.

/RTSCD

This value is for AOS/VS II only and for half–duplex modem lines only, and the system ignores it unless /HDPX is also on.

ON: Does not assert the request–to–send signal, RTS, until the carrier–detect signal, CD, is off.

OFF: The system does not check CD before asserting RTS. This is the default value.

/SFF

ON: Simulates a form feed by sending a carriage return followed by a number of NEW LINEs equal to the current number of lines per page.

OFF: Does not simulate a form feed (default).

CHARACTERISTICS (continued)

/SHR — This switch is used by certain Data General software. Do not use this characteristic because future revisions of the operating system may change its meaning or fail to support it.

/SMCD
- ON: Ignores the CD (carrier detect) signal on modem–controlled lines. You must set this switch if you set /HDPX. Also, the system ignores this switch unless you set /MOD.
- OFF: Does not ignore the CD signal on modem–controlled lines. This is the default value.

/SPO — Has no effect; formerly used with the AOS operating system.

/SRDS
- ON: Suppresses the receiver disable feature. Set this switch if your application depends on the system's honoring flow control characters even when this line is closed. Set this switch if you are attaching a device to this line that can generate X–ON/X–OFF characters at any time. These devices include some printers and asynchronous controllers.
- OFF: Does not suppress the receiver disable feature. This is the default value.

/ST
- ON: Simulates a tab stop every eighth column. This is the default value.
- OFF: Does not simulate tab stops.

/STOPBITS=n — Sets the specified number of stop bits in hardware; valid values of n are 1, 1.5, and 2.

/TCC=n — On a modem–controlled line, sets the amount of time that the system waits for a CD (carrier detect) signal after the modem is connected. The default is 40,000 milliseconds.

/TCD=n — On a modem–controlled line, sets the amount of time that the system waits for a CD (carrier detect) signal to return after it drops. The default is 5,000 milliseconds.

/TDW=n — On a modem–controlled line, sets the amount of delay time that the system imposes between a modem connect and the first I/O. The default is 10,000 milliseconds.

/THC=n — On a modem–controlled line, sets the amount of time that the system waits after a modem disconnects to let the modem settle (become stable). The default is 2,000 milliseconds.

CHARACTERISTICS (continued)

/TLT=n

The system ignores this switch unless /HDPX is also set. On a half–duplex line, sets the turnaround time. This is the amount of time that the system waits between sending the last character and dropping the RTS (Request to Send) signal. The default is 2,000 milliseconds.

/TO

ON: Enables time–outs for read and write operations.

OFF: Disables time–outs for read and write operations (default).

/TSP

ON: For AOS/VS card readers only: includes trailing spaces.

OFF: The system has no card reader or, if an AOS/VS system has one, it suppresses trailing spaces.

/UCO

ON: Converts lowercase input to uppercase on output. (Use with an uppercase–only device.)

OFF: Does not convert lowercase input. This is the default value.

/ULC

ON: Accepts both upper– and lowercase input. This is the default value.

OFF: Converts lowercase input to uppercase. Set this switch off for an application that requires only uppercase characters.

/WRP

When a program writes a line that is too long to fit on your terminal, either system software or console hardware can *wrap* the characters onto the next line.

/ON/WRP tells the system that hardware will wrap the line. /OFF/WRP tells system software to wrap the line, as governed by /CPL and /EOL. (If the /EOL switch is on and the /WRP switch is off, input that exceeds the line length is truncated.)

/WRP should be on only if the hardware will wrap at the same column as CPL (characters per line, set with the /CPL switch). The default for /WRP is on.

If you want system software to make your lines shorter, you must decrease CPL and turn /WRP off. If you want to increase your line length past the hardware margin setting (or decrease it and have the hardware wrap lines) you must adjust margins as described in the manual for your terminal.

CHARACTERISTICS (continued)

/XLT

ON: ON/XLT enables VT100 support. This is software that lets you use VT100, VT220, and other VT100–compatible terminals (including VT100 terminal emulation through an X Window System program) on your AOS/VS and AOS/VS II systems. VT100 support translates some familiar DASHER D200 cursor control keys (such as CTRL–F to move the cursor forward to the next word in SCREENEDIT mode, and so on) and function keys (SHIFT–F4 to position to the first page in the SED text editor, and so on) into equivalent VT100/VT220 commands. For International VT100 character set support, you must also use /8BT. For Kanji VT100 support, you must also use /KVT.

To run VT100–compatible terminals, your operating system must have been generated with VT100 support. For more information about VT100 support, see Appendix D.

OFF: disables VT100 support. This is the default value.

CHARACTERISTICS Example 1

```
) CHARACTERISTICS )
```

/6130/LPP=24/CPL=80/BAUD=9600/PARITY=NONE/CHARLEN=8/STOPBIT=1 /BREAK=BMOB/TCC=40000/TCD=5000/TDW=10000/THC=2000/TLT=2000 /ON/ST/EB0/ULC/WRP/IFC/OFF/SFF/EPI/8BT/SPO/RAF/RAT/RAC/NAS /OTT/EOL/UCO/MRI/FF/EB1/PM/NRM/MOD/TO/TSP/ESC/FKT/VAL /HOFC/SHR/OFC/CTD/ACC/SRDS/CALLOUT/MDUA/HDPX/SMCD/RTSCD /HIFC/AUTOBAUD/XLT

This command displays the characteristics of the terminal that is running the CLI. It is a DGC Model 6130 terminal that is set to display 24 lines per page, 80 characters per line, using a baud rate of 9600 bits per second. It accepts no parity, uses a character length of 8 with a stop bit setting of 1, clears binary mode when you press the BREAK key, and uses default timing values for the modem.

The attributes that are turned on are: simulated tab, echoing of control characters as ^, use of upper– and lowercase, line wrapping, output and input flow control. All other attributes are turned off.

CHARACTERISTICS Example 2

The following macro uses the PUSH command to move up to the next CLI environment level, and then turns page mode on before displaying the contents of the file passed to the macro as an argument. The macro then turns page mode off, and returns to the previous environment level.

```
push; prompt pop
characteristics/pm
type %1%
characteristics/off/pm
pop
```

You can use this macro by typing its name in the form

```
macroname pathname-of-file-to-type
```

CHARACTERISTICS Example 3

The following is a startup macro that is executed when a user logs on.

```
searchlist :udd:[!username] :util
defacl [!username],oware $+, +,re
characteristics/ofc/ifc/off/pm
```

The CHARACTERISTICS command identifies turns on input and output flow control, and turns off page mode. (The /ON switch could precede the /OFC and /IFC switches.)

CHARACTERISTICS Example 4

```
) CHARACTERISTICS/DEFAULT/OFF/PM  @CON3 ↵
```

This command, issued by PID 2, turns off page mode as a default setting for console 3.

CHARACTERISTICS Example 5

```
) CHARACTERISTICS/DEFAULT/MOD/MRI/BAUD=1200  @CON6 ↵
```

This command, issued by PID 2, sets new default characteristics for CON6. The new defaults take effect — that is become the current characteristics — the next time someone logs on the system from CON6 (more specifically, the next time the EXEC program enables CON6). If the terminal is already enabled, the defaults will take effect after it is disabled and enabled again.

Usually, commands such as this one are part of the system UP macro and occur in the macro before commands to EXEC enable terminals.

CHECKTERMS *Command*

Checks for the termination of a son process.

Format

CHECKTERMS *[process-ID]* (Argument applies to CLI32 only.)

Displays the process termination message from all subordinate (son) processes terminated since the last CHECKTERMS, PROCESS/BLOCK, EXECUTE or XEQ command. In CLI32, you can include a process-ID number to specify the termination message of that single process. If you include a PID, the CLI will report the termination message of that process only, and no other.

If a process has terminated abnormally (for example, it was terminated by a superior process or by an error), this command tells the CLI to display the process' termination message.

- No arguments.
- Requirement: *Standard.*
- See also: PROCESS, TERMINATE.

Why Use It?

Use the CHECKTERMS command to discover whether or not a son process has terminated and left a message. If you create a son process without blocking the CLI, the son process could terminate without your knowing about it.

You can also use CHECKTERMS to check the status of a server process with which you have established a customer-server connection (via the CONNECT command). If the server process terminates unexpectedly, you should close your end of the connection by issuing a DISCONNECT command.

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=.

CHECKTERMS Example 1

```
) PROCESS PROG1 ↵
PID:  14
...
...

) TERMINATE 14 ↵

) CHECKTERMS ↵
Process Termination, PID:  14
*Abort*

Terminated by a superior process
```

The first command creates a process called PROG1, which runs as PID 14. The next command terminates the PROG1 process. Finally, the CHECKTERMS command displays the termination message for PROG1.

CHECKTERMS Example 2

```
) PROCESS MYPROG ↵
PID:  59

) RUNTIME 59 ↵
Warning:  Attempt to access process not in hierarchy

) CHECKTERMS ↵
Process Termination, PID: 59
*Abort*
File does not exist
@INPUT
```

In this example, the first command creates a process. The second command, which tries to get runtime information about the new process, fails because the process no longer exists. The CHECKTERMS command provides information about why the process terminated.

CLASS1 *Command*

Displays or sets the Class 1 severity level.

Format

CLASS1 [*IGNORE* | *WARNING* | *ERROR* | *ABORT*]

This command either displays or sets the current severity level for Class 1 exception conditions. A Class 1 exception condition occurs when a failed command would have altered any aspect of the current CLI environment (for example, a failed DIRECTORY command).

The severity levels follow.

IGNORE	The CLI ignores the exception condition and continues executing the command or macro as best it can.
WARNING	The CLI displays a warning message and continues executing the command or macro.
ERROR	The CLI displays an error message and ceases executing the command or macro.
ABORT	The CLI terminates and returns to its father process; if the CLI is your initial process, the system logs you off.

The default Class 1 severity level is ERROR for an interactive CLI session, and ABORT for batch processing.

The CLASS1 command is useful when you want to change the error handling setting for more than one CLI command (the Class 1 global setting). For example, in a macro you might want to have the CLI continue processing commands after Class 1 errors, thus use the WARNING setting.

To set the global severity level, type CLASS1, followed by one of the four severity level arguments listed above, and press NEW LINE.

To set Class 1 error handling for a single command, append the /1=value switch, followed by the severity level option you want. For example, if you give the command

```
) DIRECTORY/1=IGNORE XXX ↵
```

and XXX is not a directory in the working directory, the CLI does not display the usual error message. For more information, type

```
) HELP *SWITCHES ↵
```

CLASS1 (continued)

- No templates.
- No argument switches.
- Requirement: *Standard.*
- See also: /1= switch (at beginning of chapter), CURRENT, CLASS2.

Why Use It?

You can use this command to change the severity level for handling all Class 1 exception conditions that subsequently occur. In some cases you may know that a program or macro will not do any harm to important data even if it encounters a Class 1 error condition. In this case you could reset the Class 1 severity level to WARNING or IGNORE to allow the macro or program to continue processing despite the error.

NOTE: The /1 switch, which is available with all CLI commands, lets you override the severity level for a Class 1 exception for that specific command. For example, if the current Class 1 setting is ERROR, the following command overrides that setting for the duration of the command:

```
) DIRECTORY/1=IGNORE ^^ )
```

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=.

/LEVEL=n — (CLI32 only.) Sets the Class 1 severity level to the setting established at 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*, replaces the current Class 1 severity level with the severity level used in the previous CLI environment. With =*n* (CLI32 only), sets the Class 1 severity level setting 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.

CLASS1 Example 1

```
) CLASS1 ↵
ERROR
```

This command displays the current severity level for Class 1 exception conditions. The ERROR setting indicates that when a Class 1 exception occurs, the CLI will display an error message and end processing of the current command or macro.

CLASS1 Example 2

```
) CLASS1 WARNING ↵
```

This command sets the severity level for a Class 1 exception condition to WARNING. Now when a Class 1 exception condition occurs, the CLI will display a warning message, but try to continue processing the command or macro.

CLASS1 Example 3

The following commands show the difference between CLASS1 ERROR, WARNING and IGNORE error handling.

) CLASS1 ↵ *ERROR*	(Check Class 1 setting.) (It is ERROR.)
) DIRECTORY XX ; TIME ↵ *Error: File does not exist, file XX*	(Type DIRECTORY and TIME commands.) (Class 1 is ERROR, so CLI displays error message and discards TIME command.)
) CLASS1 WARNING ↵) DIRECTORY XX ; TIME ↵ *Warning: File does not exist:file XX* *13:25* command.)	(Set Class 1 to WARNING.) (Repeat command.) (Since Class 1 is WARNING, CLI displays warning message but obeys TIME
) CLASS1 IGNORE ↵) DIRECTORY XX ; TIME ↵ *13:26*	(Set Class 1 to IGNORE.) (Repeat command.) (CLI omits exception message and displays time.)

CLASS2 *Command*

Displays or sets the Class 2 severity level.

Format

CLASS2 [*IGNORE* | *WARNING* | *ERROR* | *ABORT*]

This command either displays or sets the current severity level for Class 2 exception conditions. A Class 2 exception condition occurs when a failed command would have had no effect on the current CLI environment (as opposed to a Class 1 condition, which would alter some aspect of the environment). For example, a failed CREATE command causes a Class 2 error.

The severity levels follow.

IGNORE	The CLI ignores the exception condition and continues executing the command or macro as best it can.
WARNING	The CLI displays a warning message and continues executing the command or macro.
ERROR	The CLI displays an error message and ceases executing the command or macro.
ABORT	The CLI terminates and returns to its father process; if the CLI is your initial process, the system logs you off.

The CLASS2 command is useful when you want to change the error handling setting for more than one CLI command. For example, in a macro you might want to have the CLI suppress the warning message after Class 2 errors, thus use the IGNORE setting.

To set the severity level, type CLASS2, followed by one of the four severity level arguments listed above, and press NEW LINE.

To set Class 2 error handling for a single command, append the /2=value switch followed by the option you want. For example, if you give the command

```
) DELETE/2=IGNORE  SORTCOM ↵
```

and file SORTCOM does not exist, the CLI does not display the usual error message.

CLASS2 (continued)

- No templates.
- No argument switches.
- Requirement: *Standard.*
- See also: /2= switch (at beginning of chapter), CLASS1, CURRENT.

Why Use It?

Use the CLASS2 command to change the severity level for a Class 2 exception condition. You can execute a program or macro that could generate a Class 2 error and seriously affect the outcome of processing. In this case you can reset the Class 2 severity level to ERROR or ABORT to prevent the program or macro from continuing processing.

NOTE: The /2 switch, which is available with all CLI commands, lets you override the severity level for a Class 2 exception for that specific command. For example, if the current Class 2 setting is WARNING, the following command overrides that setting for the duration of the command:

```
) TYPE/2=IGNORE  MYFILE ↵
```

Another frequent use of this switch occurs when you want to delete files that might or might not exist and you don't want to see an error message about an attempt to delete a nonexistent file. An example follows.

```
) DELETE/2=IGNORE  TEMP_FILE.03_OCT+ ↵
```

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=.

/LEVEL=n — (CLI32 only.) Sets the Class 2 severity level to that established at level n.

The integer n can be absolute or relative. An unsigned integer makes nn 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*, replaces the current Class 2 severity level with the severity level used in the previous CLI environment. With *=n* (CLI32 only), sets the Class 2 severity level setting 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.

CLASS2 Example 1

) CLASS2 ↵
WARNING

This command displays the current severity level for Class 2 exception conditions. The WARNING setting indicates that when a Class 2 exception occurs, the CLI will display a warning message and try to continue processing the current command or macro.

CLASS2 Example 2

) CLASS2 IGNORE ↵

This command sets the severity level for a Class 2 exception condition to IGNORE. Now when a Class 2 exception condition occurs, the CLI will not display any message, but try to continue processing.

CLASS2 Example 3

The following macro, BOOK_UPDATE.CLI, assembles the files that make up a book, replacing any previous copy of the book. The macro uses CLASS2 to set the severity level to IGNORE. This setting allows the macro to continue execution even if some of the files that make up the book do not yet exist.

```
comment This is macro book_update.cli.
push; prompt pop
class2 ignore
delete/v  :udd1:tom:my_book
create  :udd1:tom:my_book
copy/a :udd1:tom:my_book cover
copy/a :udd1:tom:my_book preface
copy/a :udd1:tom:my_book (chapter<1 2 3 4 5 6>)
filestatus/nheader/length  :udd1:tom:my_book
pop
```

CLASS2 Example 4

The following command will delete file MY_TEMP_FILE if it exists. Otherwise, the command deletes no file and returns no message about a nonexistent file.

) DELETE/2=IGNORE MY_TEMP_FILE ↵

CLEARDEVICE *Command*

Simulates a CTRL–Q (X–ON) from a device, or sends a break character to a device.

Format

CLEARDEVICE/switch device

Depending on the switch you use, the command either simulates a CTRL–Q (X–ON) from the specified device, or sends a break character to the device. The device argument must be the name of a console line; for example, @CON12.

CLEARDEVICE is designed for system operators. They can use it for two different functions. First, it controls printers in a special way as follows. The operating system supports printers on console lines. Some of these printers use CTRL–S (X–OFF) and CTRL–Q (X–ON) characters for flow control. When they want the computer to stop transmitting, they send it CTRL–S. To resume transmission, they send it CTRL–Q.

Some printers do not send a CTRL–Q character when placed on line. If such a printer goes off line when its CTRL–S is in effect, it will never clear the CTRL–S. The printer won't operate until its CTRL–S is cleared — which you can do by simulating a CTRL–Q character with the CLEARDEVICE command. We suggest that you try this command whenever a printer attached to a console line has gone off line and won't work later when someone puts it back on line.

The second function that system operators can use CLEARDEVICE for is to transmit a break. Some communications hardware uses breaks for control. Generally, you use this function only to gain access to such hardware.

- No templates.
- No argument switches.
- Requirement: *Standard,* if you are logged onto the terminal device; otherwise *System Manager* or *PID 2.*

Why Use It?

You can use this command (with the /RXON switch) to send a CTRL–Q sequence to certain types of printers; for example, to resume printing after you load more paper. (The printer must be connected to the computer via a console line.)

You can also use CLEARDEVICE with the /SBREAK switch to send a break sequence to a device. Some types of communications hardware require you to send a break sequence before you can gain access to the device.

CLEARDEVICE 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. This section also explains the CLI32 switch /STR=.

/RXON	Simulates a CTRL–Q (X–ON) character from the device (printer).
/SBREAK	Sends a break signal to an IAC device.

CLEARDEVICE Example

You type the following command.

```
) QPRINT/QUEUE=LQP/COPIES=10  MYFILE ↵
```

The letter–quality printer begins to print. Then it exhausts its paper supply and goes off line.

The system operator replenishes the paper, puts the printer back on line; and then goes to system console and uses the CONTROL command to discover the printer devicename. The operator then uses the CLEARDEVICE command to clear the device and allow the printer to resume printing. Dialog at the system console follows.

```
) CONTROL @EXEC SPOOLSTATUS LQP ↵
LQP being processed by @CON3
From EXEC ...

) CLEARDEVICE/RXON  @CON3 ↵
```

!CLI *Pseudomacro*

Tells which CLI you are running: CLI32 or CLI16.

Format

[!CLI]

This pseudomacro returns CLI32 if your CLI process is CLI32; it returns CLI16 if your CLI process is CLI16.

Why Use It?

Use the !CLI pseudomacro when you need to know which CLI you are running — perhaps in a macro when the operations you want to perform require different treatment in each CLI.

!CLI Example

Assume you want a macro to issue one series of commands when CLI32 is running and another series of commands when CLI16 is running. You can use the following form:

```
[!EQUAL,[!CLI],CLI32]

        (cli commands to execute if CLI32 is running)

  [!ELSE]

        (cli commands to execute if CLI16 is running)

[!END]
```

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
```

CONNECT *Command*

Creates a connection with the specified server process.

Format

CONNECT { process-ID | processname | username:processname } *[...]*

This command directs the system to establish a connection between the CLI process (as a customer) and the specified server process. If the connection is made, the system displays the PID of the server process. (You use this PID to identify the server when you close the connection with the DISCONNECT command.)

A *server process* is a process that performs tasks on behalf of other processes. A process that wants to use the resources of a server process (called a *customer* of that server) must establish a connection with the server.

You must supply a process ID or a process name. If you supply a simple process name, the CLI assumes your username. You can use either process name form, but only if the process was created with the PROCESS command and /NAME=name switch.

For more information about customer–server connections, see *AOS/VS System Concepts.*

- No templates.
- No argument switches.
- Requirement: *Standard.*
- See also: DISCONNECT, CHECKTERMS, PROCESS.

Why Use It?

Use the CONNECT command if you want to use the services provided by a a server process. When you have established a connection, you can send messages to and exchange data with the server. Normally, you need this command only for customer–server programs not provided by Data General.

Monitoring the Connection

After the system establishes the customer–server connection, you can periodically use the CHECKTERMS command to see if the server process has terminated. If it has, use the DISCONNECT command to break your end of the connection.

CONNECT 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=.

/S	Stores the server's process ID (PID) in the CLI String — a useful feature if you want to use the server's PID as an argument to a command. (See the !STRING pseudomacro.)
/STRING	(CLI32 only.) Same as /S.

CONNECT Example 1

```
) PROCESS/NAME=SERVER  SERVER ↵
) CONNECT OP:SERVER ↵
Server's PID:   14
```

With the first command, username OP creates a process named SERVER o run the program SERVER.PR. The next command establishes a connection with the process OP:SERVER, which runs as PID 14.

CONNECT Example 2

```
) CONNECT  27 ↵
Server's PID:   27
...
...
) CHECKTERMS ↵
Process termination, PID: 27
*Abort*

Terminated by a superior process

) DISCONNECT 27 ↵
```

The first command establishes a connection with a server running as PID 27. Later, a CHECKTERMS command reports that the server has been terminated. The DISCONNECT command breaks the customer end of the connection.

CONNECT Example 3

```
) CONNECT/S  22 ↵      (Connects the user's process with server
SERVER'S PID: 22       process 22 and stores PID in the CLI String.)

) STRING ↵             (Displays the current value of the String.)
22
```

!CONSOLE *Pseudomacro*

Expands to a console name or a batch job queue name.

Format

[!CONSOLE]

For a process running under EXEC, this pseudomacro returns the name of the process's console; for a batch process, the pseudomacro returns the name of the queue.

- No arguments.
- No macro name switches.
- Requirement: *Standard.*
- See also: !LOGON, which you can use to determine whether a process is batch or interactive.

Why Use It?

This pseudomacro is useful within macros when you want to use either the name of the console or the batch queue as an argument to another command.

!CONSOLE Example 1

```
) WRITE My terminal name is @[!CONSOLE]. )
My terminal name is @CON6.
```

This command displays a message that reports the current console name.

!CONSOLE Example 2

(within a macro)

```
...
send @[!console] Are you ready?
...
```

This macro statement sends the message *Are you ready?* to the current username at the current terminal.

CONTROL *Command*

Sends a control message to a process.

Format

CONTROL ipc-port message ...

This command sends a message string to process via the specified IPC (Interprocess Communication) port.

Certain processes can exchange messages with other processes via the Interprocess Communications (IPC) facility. To use this facility, the process sends its message over a communications path (called a *port*), which is identified by a unique port number.

To send an IPC message via the CLI, you specify the port by using the pathname of the process's IPC file in the :PER directory. So, to send a message to the EXEC process, for example, you specify the port as @EXEC.

- No templates.
- No argument switches.
- Requirement: *Username OP* or *System Manager* privilege.

Why Use It?

If you are the system operator, you use this command to send control messages to processes such as EXEC, INFOS II, and XTS (XODIAC Transport Server).

For detailed information about the practical uses of this command, see the manual *Managing AOS/VS and AOS/VS II* or the reference manual for the product you are using.

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=.

/I — Accepts as messages any subsequent lines typed in at the keyboard (@INPUT). It sends each line as a separate IPC message. To end the last message, type a close parenthesis,), on the last line, and press NEW LINE.

Note that the CLI does not interpret pseudomacros given as arguments with the CONTROL/I switch; it ignores them.

/INPUT — (CLI32 only.) Same as /I.

CONTROL (continued)

/M

Takes the data from the specified macro file and uses it as the IPC messages. The system sends each line of the macro as a separate IPC. The only argument allowed to the CONTROL command is an IPC port. To end the last message, type a close parenthesis,), on the last line, and press NEW LINE. For example, macro XSTATUS.CLI contains

```
CONTROL/M @EXEC
SPOOLSTATUS
STATUS
)
```

If you type XSTATUS and press NEW LINE, EXEC will execute the SPOOLSTATUS and STATUS commands. (You must have username OP or System Manager privilege to use EXEC commands.)

Note that you cannot use pseudomacros as arguments within a macro you call with the CONTROL/M switch.

/MACRO

(CLI32 only.) Same as /M.

CONTROL Example 1

```
) CONTROL @EXEC RESTART @LPB ↵
```

This command directs the EXEC process to restart output on the line printer — perhaps after a paper jam was corrected.

CONTROL Example 2

```
) CONTROL @EXEC ENABLE @CON23 ↵
```

This command directs the EXEC process to enable console 23.

CONTROL Example 3

```
) CONTROL @FTA START ↵
```

This command starts the FTA process.

CONVERT *Utility*

Converts an RDOS .RB file to an AOS/VS or AOS/VS II .OB file.

Format

XEQ CONVERT pathname*[.RB]*

RDOS (the Real–time Disk Operating System) is another Data General operating system. RDOS supports a relocatable binary (.RB) module, which is not compatible with AOS/VS or AOS/VS II. The CONVERT utility converts a .RB file to an AOS/VS or AOS/VS II object file (.OB). You must supply the pathname of the .RB file that you want to convert; you can omit the .RB filename suffix.

This utility creates a new .OB file with the same root name as the .RB file. The original .RB file remains unchanged.

NOTE: This utility cannot convert RDOS system calls to AOS/VS or AOS/VS II system calls. You must rewrite any RDOS system calls and then reassemble the program before converting it for use on AOS/VS or AOS/VS II systems.

- Does not accept templates.
- Requirement: *Standard* (Execute access to program file in :UTIL).

Why Use It?

Use the CONVERT utility to build an object binary file which you can link into a program to run on an AOS/VS or AOS/VS II system from one or more relocatable binary files that were compiled on an RDOS system. (The .RB files are not compatible with AOS/VS or AOS/VS II systems.)

CONVERT Example

```
) XEQ CONVERT PLUS24 ↵
PLUS24.RB
```

This command converts an RDOS relocatable binary file called PLUS24.RB to an object file that is compatible with AOS/VS and AOS/VS II systems. (The utility displays the name of the .RB file when it opens the file.)

The output of this utility is an object file named PLUS24.OB. You can now include the .OB file in a Link command line to build a program that will run on an AOS/VS or AOS/VS II system.

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.

COPY (continued)

/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.

CPIO_VS (continued)

/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
```

CREATE *Command*

Creates a file.

Format

CREATE { pathname
/LINK pathname link–resolution–pathname }

This command creates a disk file, which you specify by pathname. The type of file depends on your use of command switches. If you create a link file, you must also use the /LINK switch and specify the pathname of the file to which the link refers.

- No templates.
- No argument switches.
- Requirement: *Standard.*
- See also: COPY, MOVE, PERMANENCE, ACL.

Why Use It?

Use the CREATE command to create a disk file, such as a directory, link, or other kind of file.

You may want to create an empty file before performing an operation that expects a file to exist. For example, a macro might use a COPY/A command to append a specified file to a master file, but the command will cause an error if the master file does not exist. Before issuing the COPY/A command, you could use CREATE to establish the master file.

You also use this command to create a file with a specific record format (such as data–sensitive).

Use this command if you want to create a file and specify its file type (such as the user–defined file types shown in Table 2–8).

With the /ELEMENTSIZE= and /INDEX= switches, you can use this command to create a large contiguous file. Many data management programs require such a file.

CREATE 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=.

/DATASENSITIVE	Creates a file with a data–sensitive record format.
/DIRECTORY	Creates a directory of type DIR; this type of directory grows as needed. If you use the /MAXSIZE=n switch, the system creates a control point directory with maximum size n blocks.

CREATE (continued)

/DYNAMIC — Creates a file with a dynamic record format.

/ELEMENTSIZE=value — Sets the file element size. A file element is a set of contiguous 512–byte disk blocks (usually 4 blocks); it represents the minimum amount of space by which a file can grow. An advantage of small elements is that, as a file grows, the system can more easily find contiguous space for new elements. An advantage to a large element size is that, with an index level of 0, access times tend to be short since there is no need to search a file index. An example switch to specify a large element size is /ELEMENTSIZE=32636/INDEX=0.

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 disk blocks or a value chosen during AOS/VS system generation.

In AOS/VS II, there are two types of file elements: primary and secondary. With CLI32, You can specify the size of each type, and the number of primary elements, using the form

/ELEMENTSIZE=primary–size:secondary–size:primary_quantity

For example,

) CREATE/ELEMENTSIZE=32768:4:8 CONTIGUOUS_FILE ↵

This creates a file with a primary element size of 32768, secondary element size of 4, and 8 primary elements. If you omit the primary element size and/or quantity (including only a colon for the omitted value), the system uses the LDU default values. For example, /ELEMENTSIZE=:8: specifies the defaults for all but secondary element size (8).

/FIXED=n — Creates the file with a fixed–length record format of n bytes.

/HASHFRAMESIZE=n — For AOS/VS II, this switch has no effect. For AOS/VS, it sets the directory's hash–frame size (the unit into which the system divides the directory for file access) to value n. The default size (7) is suitable for directories containing about 140 files. For optimum access, divide the number of files by 20 and take the resulting prime number or the next higher one, up to 157. For example, the best hash–frame size for 300 files is 17.

CREATE (continued)

/INPUT

Inserts text typed at the terminal (@INPUT) as contents to the file. To end input, type a close parenthesis,), on a line, without any following arguments. Then press NEW LINE. Does not interpret pseudomacros.

/I

(CLI32 only.) Same as /INPUT.

/INDEXLEVELS=n

Sets the maximum number of index levels for the file to n (where n is from 0 through 3; the default number is 3). The system creates the file with 0 index levels and later creates other levels as needed, up to the maximum you specify (no more than 3). An index level of 0 limits a file to 1 (contiguous) element; a level of 1 limits it to 512/m elements; a level of 2 limits it to (number–of–elements–in–level–1) squared; and so on. (The m is the number of disk blocks in an element; the default size is 4 but you can select another number with /ELEMENTSIZE=.)

With CLI32 and AOS/VS II, you can also specify the index element size in disk blocks, by using the format n:index–elementsize for n. For example, CREATE/INDEXLEVELS=3:2 MYFILE sets the maximum number of index levels to 3 and the element size of each index block to 2 disk blocks. If you do not specify the element size, it is created as the default LDU index element size, 1.

/LINK

Creates a link file (type LNK) that resolves to the resolution pathname argument. For example, the following command creates a link file named SED.PR to resolution file SED.PR in directory :UTIL.

```
) CREATE/LINK  SED.PR  :UTIL:SED.PR ↵
```

/MACRO

Takes the contents of the file from the macro body that follows. To end input from the macro, type a close parenthesis,) and press NEW LINE. Does not interpret pseudomacros within the macro.

/M

(CLI32 only.) Same as /MACRO.

/MAXSIZE=n

Creates a control point directory (CPD) of size n (number of 512–byte disk blocks). For example, to create directory MYDIR with a maximum size of 10,000 blocks:

```
) CREATE/MAXSIZE=10000 MYDIR ↵
```

CREATE (continued)

/TYPE=typecode	Creates a file of the type specified by type code, where the type code is an integer in the range 64 through 255. (See Table 2–8 for a listing of file types by number.)
/VARIABLE	Creates the file with variable record format.

CREATE Example 1

```
) CREATE/DIR PROGRAMS ↵
```

This command creates a directory file called PROGRAMS.

CREATE Example 2

```
) CREATE/I PROGRAMS.CLI ↵
)) DIRECTORY :UDD:TOM:PROGRAMS ↵
)) DIRECTORY ↵
)) ) ↵
```

The first command creates a macro file called PROGRAMS.CLI. The /I switch causes subsequent input to be inserted into that file. The next two commands set the working directory to PROGRAMS, and display that directory's pathname. The single right parenthesis ends input.

```
) PROGRAMS ↵
:UDD:TOM:PROGRAMS
```

This command executes the macro just created.

CREATE Example 3

```
) CREATE/LINK  PROGS  :UDD:TOM:PROGRAMS.DIR ↵
```

This command creates a link file called PROGS in the working directory. PROGS refers to the file :UDD:TOM:PROGRAMS.DIR.

CREATE Example 4

```
) CREATE/LINK  JAN_90  :UDD1:SALES:ACCOUNTS:JAN_90 ↵
```

This command creates a link file JAN_90 in the working directory to resolution file :UDD1:SALES:ACCOUNTS:JAN90. The link provides easy access, by filename JAN_90, to the resolution file; as, for example, via

```
) TYPE  JAN_90 ↵
```

CREATE Example 5

```
) CREATE/DIRECTORY/MAXSIZE=5000  SCHEDULES ↵
```

This command creates a control point directory, and specifies its maximum size as 5000 blocks (each block contains 512 bytes).

CURRENT *Command*

Displays the settings in the current CLI environment level.

Format

CURRENT

This command displays the values of all settings that apply to the current CLI environment level.

- No arguments.
- Requirement: *Standard.*
- See also: PREVIOUS, POP, PUSH, LEVEL.

Why Use It?

Use the CURRENT command to get a complete picture of the current environment level. Before using an environment–dependent value, you might want to use the CURRENT command to get an overview of all the settings that are in effect.

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=.

/LEVEL=n (CLI32 only.) Displays the settings for the environment on 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.

/SAVEALL=pathname (CLI32 only.) Saves all commands needed to create the current environment (or the environment specified in /LEVEL=n) in the file indicated by pathname. You can simply run that pathname (it's a CLI macro) to recreate the environment.

CURRENT Example 1

```
) CURRENT ↵
LEVEL              0
SUPERUSER          OFF
SUPERPROCESS       OFF
SYSTEMMANAGER      OFF                                  (CLI32 only)
SCREENEDIT         ON
SQUEEZE            OFF
CMD # IN PREFIX    OFF                                  (CLI32 only)
CLASS1             ERROR
CLASS2             WARNING
TRACE
VARIABLES          0    0      0      0      0
                   0    0      0      0      0
LISTFILE           @LIST
DATAFILE           @DATA
LOGFILE
DIRECTORY          :UDD1:TOM:CLI:COMMANDS
GROUPLIST                                               (CLI32 only)
SEARCHLIST         :UDD1:TOM,:UDD1:TOM:MACROS:UTIL,:
DEFACL             TOM,OWARE $+,  +,E
STRING
PROMPT
PREFIX                                                  (CLI32 only)

CHARACTERISTICS      /6130/LPP=24/CPL=80/BAUD=9600/PARITY=NONE
                     /CHARLEN=8/STOPBITS=1/BREAK=BMOB/ON/ST
                     /EB0/ULC/WRP/OFC/IFC/OFF/SFF/EPI/8BT/SPO/RAF
                     /RAT/RAC/NAS/OTT/EOL/UCO/MRI/FF/EB1/PM
                     /NRM/MOD/TO/TSP/PBN/ESC/FKT/NNL/SHR/CALLOUT
                     /MDUA/HDPX/SMCD/RTSCD/HIFC/G1G0/DKHW/NLX
```

CURRENT Example 2

```
) PUSH ↵
) DIR  :UDD:XDIR:APPLICATIONS ↵
) CHARACTERISTICS/ON/PM/CHARLEN=8 ↵
) CURRENT/SAVEALL=LEVEL_2_ENV.CLI ↵
```

This example, for CLI32 only, saves the current environment in file LEVEL_2.CLI; the person can later restore that environment by running the macro.

DATAFILE *Command*

Displays or sets the current data file pathname.

Format

DATAFILE *[pathname]*

This command either displays the pathname of the current data file, or lets you designate an existing file as the current data file.

The default setting is @DATA, which means that no file is currently designated as the data file. You must designate a data file before you execute a program that tries to open and read the generic @DATA file.

- No templates.
- No argument switches.
- Requirement: *Standard.*
- See also: !DATAFILE, LISTFILE.

Why Use It?

Use the DATAFILE command to designate an existing file as the current environment's data file. Before running a program that uses the generic @DATA file for input, you can assign a specific file to be used whenever the program refers to the @DATA file. In this way you can run the same program several times, but use a different data file each time. (See Example 2.)

At any time you can display the pathname of the current data file by entering the command without a pathname argument.

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=.

/G	Sets the filename to @DATA. (Omit the pathname argument.) Do not use this switch together with /K.
/GENERIC	(CLI32 only.) Same as /G.
/K	Sets DATAFILE to a null string. (Omit the pathname argument.) Do not use this switch together with /G.
/KILL	(CLI32 only.) Same as /K.

DATAFILE (continued)

/LEVEL=n — (CLI32 only.) Sets the data file to the one established at level n. (Omit the pathname argument.)

The integer n can be absolute or relative. An unsigned integer makes nn 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 DATAFILE to the pathname used in the previous environment level. With *=n* (CLI32 only), sets the data file to that 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.

DATAFILE Example 1

```
) DATAFILE ↵
@DATA
```

This command shows the current data file assignment, which is @DATA.

DATAFILE Example 2

```
) DATAFILE  SURVEY.JULY31 ↵

) XEQ SURVEY_REPORT ↵
Creating a report with data collected for 31 July ...

Report complete.

) DATAFILE  SURVEY.AUGUST01 ↵

) XEQ  SURVEY_REPORT ↵
Creating a report with data collected for 01 August ...

Report complete.
```

The previous sequence of commands shows how you can associate a specific data file to be used by a program that opens and reads the generic @DATA file. The same program can be run without modification, but using different input files.

DATAFILE (continued)

The source code for program SURVEY_REPORT contains references to file @DATA and no references to files SURVEY.JULY31, etc. If the program is written in FORTRAN 77, a part of file SURVEY_REPORT.F77 might contain the following statements (where a statement beginning with C is a FORTRAN 77 comment statement):

```
C       Open the input file.  The actual file is supplied at runtime as the
C       argument to a CLI DATAFILE command.  This command occurs
C       before the CLI command that executes this program.
C
        OPEN (2, FILE='@DATA', IOINTENT='INPUT', RECFM='DATASENSITIVE')

C       The I/O intent is to only read data from the file; its record format
C       is data sensitive.
```

!DATAFILE *Pseudomacro*

Expands to the full pathname of the current data file.

Format

[!DATAFILE]

This pseudomacro returns the full pathname of the current data file.

NOTE: The data file may differ from one CLI environment to another. If you change environments via the PUSH or POP command, the pseudomacro may return different pathnames.

- No arguments.
- Accepts a macro name switch (described later).
- Requirement: *Standard.*
- See also: DATAFILE.

Why Use It?

Use the !DATAFILE pseudomacro to substitute the full pathname of the current data file within a CLI command argument. A macro that executes a program could write out statistical information following program execution. The macro could use !DATAFILE to include the name of the data file that the program used.

A macro can test the value of !DATAFILE to determine whether or not a data file is currently assigned. If the returned value is @DATA, the macro could assign a data file before executing a program that expects a data file.

Macro Name Switch

/LEVEL=n — (CLI32 only.) Sets the data file to the one established at 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*, expands to the full pathname of the previous CLI environment's data file. With =*n* (CLI32 only), expands to the full pathname of the specified environment level. The *n* specifies the number of levels above the current level (toward 0).

/PREVIOUS*[=n]* — (CLI32 only.) Same as /P.

!DATAFILE Example 1

(within a macro)

```
...
copy/a  %1%  [!datafile]
...
```

This macro statement appends the contents of the current data file to the file specified by the first argument passed to the macro.

!DATAFILE Example 2

```
) XEQ MYPROGRAM  [!DATAFILE] ↵
```

This command executes MYPROGRAM and provides the current data file as an argument to the program.

!DATAFILE Example 3

```
comment This is macro DISPLAY_DATAFILE.CLI.
[!equal,[!datafile],@DATA]
        write No data file is assigned.  Use DATAFILE to designate one.
[!else]
        write The current data file is [!datafile].
[!end]
```

This macro tests the current data file assignment.

!DATAFILE Example 4

```
) WRITE THE CURRENT DATA FILE IS [!DATAFILE] ↵
THE CURRENT DATA FILE IS @DATA
```

The CLI evaluates [!DATAFILE] as the current data file (in this case the generic @DATA).

```
) PUSH ↵
) DATAFILE :UDD:USER:WORK ↵

) WRITE Now the current data file is [!DATAFILE] ↵
Now the current data file is :UDD:USER:WORK

) WRITE [!DATAFILE/P] ↵
@DATA
```

We then change the environment, and set :UDD:USER:WORK as the data file for that environment. The CLI now evaluates [!DATAFILE] as :UDD:USER:WORK for the current environment, and with the /P switch, as @DATA for the previous environment.

DATE *Command*

Displays or sets the current system date.

Format

DATE *[dd–mon–yy]*

This command either displays the current system date, or lets the system operator or system manager change the system date. To specify the date, use the format

dd–mon–yy	(hyphens required)— For example, 24–NOV–90
dd	Integer to indicate the day of the month.
mon	Three–letter abbreviation for the month (JAN, FEB, MAR, ...)
yy	Integer — the last 2 digits of the year.

- No templates.
- No argument switches.
- Requirement: *Standard* to display; *PID 2* or *System Manager* to set.
- See also: !DATE, TIME, !TIME.

Why Use It?

You can use the DATE command to display the current date.

If your process is PID 2 or has System Manager mode turned on (CLI32 only), you can use this command to set the current date or to correct the date if it was entered incorrectly at system startup.

If your computer system has a boot clock (time–of–day clock), this command also sets the boot clock.

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=.

DATE Example 1

```
) DATE ↵
12-JUN-90
```

This command displays the current date.

DATE Example 2

```
Sm) DATE 26-NOV-90 ↵
```

This process (PID 2 or with System Manager privilege turned on) sets the date to November 26, 1990.

DATE Example 2

```
) PROMPT DATE TIME ↵
31-JUL-90
13:45:06
```

You can use the DATE command as an argument to the PROMPT command so that the system will display the date whenever the CLI prompts you for input.

!DATE *Pseudomacro*

Expands to the current system date.

Format

[!DATE *date–or–integer*[1]] 1 CLI32 only.

This pseudomacro returns the current system date in dd–mon–yy format, (04–NOV–90, for example). Note that the day is always represented by two digits.

With CLI32, you can insert a date or integer as an argument; if you insert a date with the /NUMERIC switch, the CLI will convert it to the integer number of days after January 1, 1968. If you insert an integer as an argument, the CLI will translate it to the corresponding date after January 1, 1968.

- Argument allowed in CLI32 only.
- Macro name switch in CLI32 only.
- Requirement: *Standard.*
- See also: DATE, !EXPLODE (macro TODAY.CLI), !TIME, !VAR.

Why Use It?

Use the !DATE pseudomacro (and/or !TIME pseudomacro) to include the system date within a CLI command argument. Within a macro, for example, you can include the current date in a message that the macro displays. A macro can also use the !DATE pseudomacro to specify actions that depend on the current system date. With CLI32, you can use the /NUMERIC switch to meter the passage of time.

Macro Name Switch

/NUMERIC	(CLI32 only.) Displays the date as a number of days since January 1, 1968; or, if you include the date, converts the date you specify to the integer number of days after January 1, 1968.

!DATE Example 1

```
) WRITE Today's date is [!date] ↵
Today's date is 12–JUN–90.
) FILESTATUS/SORT/AFTER/TLM=[!DATE] ↵
...(Displays files created or modified today)...
```

The first command displays the date; the second displays the names of all files that were created or modified today.

!DATE Example 2

```
) WRITE  Today is [!DATE/NUMERIC] days after 1/1/68.↵
Today is 8026 days after 1/1/68.
) WRITE  January 1 1981 was [!DATE/NUMERIC 1-JAN-81] days after 1/1/68.↵
January 1 1981 was 4750 days after 1/1/68.↵
```

For CLI32 only, these commands display the number of days that have passed since January 1, 1968, and then display the number of days that January 1, 1981 occurred after January 1, 1968.

DEASSIGN *Command*

Releases a previously assigned character device.

Format

DEASSIGN devicename ...

This command releases the named character device(s), which had been reserved for your exclusive use.

- Accepts templates.
- No argument switches.
- Requirement: *Standard* (but assigning a device requires PID 2).
- See also: ASSIGN.

Why Use It?

Use the DEASSIGN command to release a previously reserved character device so that other processes can use 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 switch /STR.

DEASSIGN Example

```
) ASSIGN @CON64 ↵
...
...
) DEASSIGN @CON64 ↵
```

The first command assigns terminal 64 for your exclusive use. After using the terminal, you issue the DEASSIGN command to release it for use by others.

DEBUG *Command*

Executes the specified program and enters the assembly language debugger.

Format

DEBUG pathname *[argument] [...]*

This command executes the program file specified in pathname, and starts the program as a subordinate process of the debugger. The CLI first tries to run pathname.PR; if it does not find pathname.PR, it tries to run pathname. Next, the system passes the arguments to the new process. This process can gain access to the arguments through the ?GTMES system call.

For example, the command

) DEBUG MY_PROG FILE07 49 ↵

tells the system to execute the debugger and execute program file MY_PROG.PR as a subordinate process of the debugger. Program MY_PROG can issue system call ?GTMES to get the arguments FILE_07 and 49 for processing by the program.

- No templates.
- Accepts argument switches.
- Requirement: *Standard.*
- See also: SWAT; for debugger information, the *AOS/VS Debugger and File Editor User's Manual*; for the ?GTMES system call, the *AOS/VS, AOS/VS II, and AOS/RT32 System Call Dictionary, ?A through ?Q*; for processes, the PROCESS command or *AOS/VS System Concepts;* for arguments, the user manual for the program (if any) manual

Why Use It?

Use the DEBUG command to examine a program at the assembly language level, to check its logic, or to trace the cause of an error.

DEBUG 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.

/I	Takes input for the program you are debugging (not for the debugger itself) from subsequent lines typed at the keyboard (@INPUT). To end input, type a close parenthesis,), on a line (without any following arguments), and then press NEW LINE. The CLI will not interpret pseudomacros within /I.
/INPUT	(CLI32 only.) Same as /I.

DEBUG (continued)

/M	Takes input for the program you are debugging (not for the debugger itself) from the macro body that follows. The CLI will not interpret pseudomacros in the macro. To end the input from the macro to the program, type a close parenthesis,), without any following arguments.
/MACRO	(CLI32 only.) Same as /M.
/S	Places the program's termination message in the CLI String.
/STRING	(CLI32 only.) Same as /S.

DEBUG Example

```
) DEBUG MYPROGRAM ↵

AOS/VS User Debugger– Rev. XX

00000000000 00000000000 00000000000 00000000000
00000000000

_$Z
```

This command calls the assembly language debugger to debug a program called MYPROGRAM. The command ESC Z (which appears as $Z) ends the debugging session.

!DECIMAL *Pseudomacro*

Expands to the decimal equivalent of an octal number.

Format

[!DECIMAL octal–number *[...]*[1]]

[1] Multiple arguments available in CLI32 only.

This pseudomacro returns the decimal equivalent of an octal integer. The argument must be in the range 0 through 37,777,777,777 (resulting in a decimal value in the range 0 through 4,294,967,295). With CLI32, you can specify multiple arguments.

- No templates.
- No argument switches.
- No macro name switches.
- Requirement: *Standard.*
- See also: !OCTAL.

Why Use It?

Use the !DECIMAL pseudomacro to convert an octal integer to a decimal value, and use that value as an argument to a CLI command.

!DECIMAL Example 1

This command displays the decimal equivalent of the octal value 1750.

```
) WRITE [!DECIMAL 1750] ↵
1000
```

!DECIMAL Example 2

The macro DECIMAL.CLI contains the following line:

```
write Octal %1% is [!decimal %1%] decimal.
```

The macro takes an octal integer as an argument, and then displays the decimal equivalent. For example,

```
) DECIMAL 141 ↵
Octal 141 is 97 decimal.
```

See also the CALCULATOR macro in Chapter 4.

DEFACL *Command*

Displays or sets your default access control list.

Format

DEFACL pathname [*username, access-types [...]*
username[:groupname][,...],access-types [...][1]]

[1] CLI32 with AOS/VS II only.

This command either displays your current default access control list, or lets you set it. When you create a file, the system automatically uses your default ACL as the new file's access control list.

If you omit arguments, the system displays your default access control list.

If you specify an ACL, the command assigns that ACL as the default ACL. 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). Your default ACL should give your username all access (for example, DEFACL [!USERNAME],OWARE. However, you can specify more than one username–access pair. You can separate the username from the access types by a comma or space; for example DEFACL 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

With AOS/VS II and CLI32, a username can include a groupname, on the form username:groupname,access (for example, DEFACL MYFILE JONB,OWARE +:PROJECTX,WARE +,RE). For group access to work, there must be a file named groupname in directory :GROUPS, and the username must be defined in that file. Groups are further explained in Chapter 2 and in *Managing AOS/VS and AOS/VS II.*)

- No templates.
- No argument switches.
- Requirement: *Standard.*
- See also: ACL, CREATE, !DEFACL.

DEFACL (continued)

Why Use It?

You can use the DEFACL command to display your default access control list if you are unsure about the ACL the system uses for your new files.

Or you can use this command to change your default ACL. If you normally allow others working on the same project to gain access to your files, you may want to update your default ACL to provide access to members of your project team.

You can include the DEFACL command in your startup macro to set your default ACL each time you log on.

With AOS/VS II and CLI32, when you join a username group (GROUPLIST command), you can use the DEFACL command to set the default ACL to give group members access to files you create.

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=.

/D	Uses the standard default access control list: your–username, OWARE. (No arguments allowed.)
/DEFAULT	(CLI32 only.) Same as /D.
/K	Kills the ACL and assigns a null ACL. No one can access a file that has a null ACL (unless he or she has Superuser mode on). However, anyone with Write access to the parent directory can change this ACL. (No arguments allowed.)
/KILL	(CLI32 only.) Same as /K.
/LEVEL=n	(CLI32 only.) Sets the default ACL to that established at 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 default access control list to the previous environment's default ACL. With =*n* (CLI32 only), sets the default access control list to the specified environment level. The *n* specifies the number of levels above the current level (toward 0).
/PREVIOUS*[=n]*	(CLI32 only.) Same as /P.

DEFACL Example 1

```
) DEFACL ↵
MARY,OWARE

) CREATE NEWFILE ↵
) ACL/V NEWFILE ↵
=NEWFILE                    MARY,OWARE
```

The first command reports the current default access control list. The second command creates a file. The third command shows that the default ACL was used for the new file.

DEFACL Example 2

```
) DEFACl ↵
MARY,OWARE

) DEFACL [!DEFACL] +,RE ↵
) DEFACL ↵
MARY,OWARE +,RE
```

The first command displays the current default ACL. The second command changes the default ACL to allow other users Read and Execute access. (The !DEFACL pseudomacro represents the current default ACL.) The last command confirms the new default ACL.

DEFACL Example 3

```
) DEFACL [!USERNAME],OWARE  +:MARK_II,WARE  +,RE ↵
) DEFACL ↵
JKM,OWARE +:MARK_II,WARE +,RE
```

For AOS/VS II and CLI32 only, the default ACL bestowed by this DEFACL command gives user JKM all access. It gives any user who is a member of group MARK_II WARE access. And it gives all other users RE access.

!DEFACL *Pseudomacro*

Expands to the current default access control list.

Format

[!DEFACL]

This pseudomacro represents the current default access control list for the CLI user.

- No arguments.
- Accepts a macro name switch (described later).
- Requirement: *Standard.*
- See also: DEFACL.

Why Use It?

The !DEFACL pseudomacro is useful as a shorthand method for including your default ACL in a command. Example 1 shows how you can use this pseudomacro with the DEFACL command to add access control information to your existing default ACL.

Macro Name Switches

/LEVEL=n — (CLI32 only.) Expands to the default ACL established at level n.

The integer n can be absolute or relative. An unsigned integer makes nn 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*, expands to the default ACL in the previous environment level. With =*n* (CLI32 only), expands to the default ACL 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.

!DEFACL Example 1

```
) DEFACL [!DEFACL] $GUEST,E ↵
```

This command changes the default access control list setting by adding the username/access type of GUEST,E to the current default ACL.

!DEFACL Example 2

```
) ACL MYFILE [!DEFACL] MARK,WRE ↵
```

This command sets the access control list for MYFILE to the current default ACL plus MARK,WRE. Using the pseudomacro frees you from having to type the username–template/access–type string of the default ACL.

DELETE *Command*

Deletes one or more files.

Format

DELETE pathname *[...]*

This command deletes the specified file(s). If you delete a directory, you delete its contents as well. (The system does not let you delete a directory that contains subdirectories unless you use the # template.)

To delete a file, you must have Write access to its parent directory or Owner access to the file itself, or you must have Superuser turned on.

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

Why Use It?

Use the DELETE command to permanently remove a file from the directory that contains it. You may want to discard old data to make space available for new data. You can use this command also to delete files that you have copied to a different location.

You can also use this command to delete temporary files (usually identified by the suffix .TM or .TMP) that are no longer needed.

If a batch job you submitted with QBATCH aborts or if you cancel it, (via the QCANCEL command), you can use DELETE to remove the batch input file from your directory. To learn the names of these files, type FILESTATUS ?+.CLI.+.JOB and press NEW LINE in the directory from which you submitted the job.

A Word of Caution

After you delete a file, there is no way to retrieve it unless you have stored a copy under another name, in a different directory, or on an external medium (such as a magnetic tape).

Be careful when using a template argument with the DELETE command. To help guard against deleting files that you want to keep, include the /C (confirm) switch, which lets you confirm or cancel the deletion of each file that matches the template. Or you can list files that would be deleted by using the FILESTATUS command (instead of DELETE); then if you really want to delete these files, substitute DELETE for FILESTATUS.

DELETE (continued)

You can also preview the effect of a DELETE command by using the WRITE command. See Example 3.

Another way to protect your files from accidental deletion is to turn on permanence for those files. You may also want to turn permanence on for the directory that contains the files. Under AOS/VS, you can delete a nonpermanent directory that holds permanent files. AOS/VS II will not let you delete any directory that holds permanent files; instead it will display the error message *Directory delete error.*

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–mmm–yy:hh:mm:ss), date (dd–mmm–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–mmm–yy:hh:mm:ss), date (dd–mmm–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.
/C	Displays the name of each file to be deleted, and then prompt for confirmation. (To delete the file, enter Y; to retain it, enter N.)
/CONFIRM	(CLI32 only.) Same as /C.
/COUNT	(CLI32 only.) Counts the number of files this command processes.
/SORT	(CLI32 only.) Sorts alphabetically the filenames deleted. To see the names, you must also include /C, /V, or both.

DELETE (continued)

/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

```
) DELETE/TYPE=\CPD #:PROJ.- ↵
```

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

```
) DELETE/TRAVERSE=\CPD #:PROJ.- ↵
```

give expected results.

/TYPE=typecode — (CLI32 only.) Specifies one or more types of files to process. Valid type codes appear in Table 2–8.

/V — Displays the name of each file that is deleted.

/VERIFY — (CLI32 only.) Same as /V.

DELETE Example 1

```
) MOVE/V :UDD:COMMON FINALSPEC ↵
) DELETE/V FINALSPEC ↵
Deleted FINALSPEC
```

The first command moves a copy of the file FINALSPEC from your working directory to the directory :UDD:COMMON. After the first command executes, the user issues the second command to delete the original copy from his working directory.

DELETE Example 2

```
) DELETE/C/V F+ ↵

=FEE? Y↵
Deleted =FEE

=FI? Y↵
Deleted =FI

=FILESYSTEM_STDS n ↵
File not deleted =FILESYSTEM_STDS

=FUM? Y↵
Deleted =FUM
```

This command requests the deletion of all files in the working directory that begin with the letter F. The /C switch requests the CLI to prompt for confirmation before deleting a file, and the /V switch asks the CLI to verify each deletion.

DELETE Example 3

```
) WRITE DELETE/V  [!FILENAMES TEMP+] ↵
DELETE =TEMPA =TEMP14 =TEMPC =TEMP3 =TEMPB

) DELETE/V [!FILENAMES TEMP+] ↵
deleted =TEMPA
deleted =TEMP14
deleted =TEMPC
deleted =TEMP3
deleted =TEMPB
```

The first command displays names of files *without* deleting them, thus letting you preview what files the command, without WRITE, would delete. The second command actually deletes them. (You can create the second command from the first by typing CTRL–A and then using the Screenedit control characters.) An equivalent of the second command is DELETE/V TEMP+.

DELETE Example 4

```
) DELETE/C/V/BEFORE/TLM=1–JAN–90  + ↵
```

In CLI32 only, this command tells the CLI to select — and prompt for deletion of — each file last modified before January 1, 1991.

DIRECTORY *Command*

Displays or sets the working directory.

Format

DIRECTORY *[pathname]*

This command displays the pathname of your working directory, or lets you set your working directory to a specified directory. The pathname must specify a valid (existing) directory. If you specify a filename rather than a directory name (a file of a type other than CPD, DIR, or LDU), the command will not work.

- No templates (but you can use pathname prefix characters, such as ^).
- No argument switches.
- Requirement: *Standard.*
- See also: CREATE, FILESTATUS, SEARCHLIST.

Why Use It?

You will use this command often to display the pathname of your working directory. This shows your location in the directory tree.

You also use this command to move from one directory to another. You might want to do this if you need to work with one or more files in a specific directory. You can refer to files in your working directory without using a full pathname; the system looks in your working directory before scanning the directories on your search list.

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=.

/I	Sets the working directory to the initial working directory.
/INITIAL	(CLI32 only.) Same as /I.
/I pathname	Sets the working directory to the directory specified by pathname, which starts from the initial working directory. In other words, pathname is relative to the initial working directory. See the last part of Example 1.
/INITIAL pathname	(CLI32 only.) Same as /I pathname.

DIRECTORY (continued)

/LEVEL=n — (CLI32 only.) Sets the working directory 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 working directory to the previous environment's working directory. With =*n* (CLI32 only), sets the working directory to the working directory of the specified environment level. The *n* specifies the number of levels above the current level (toward 0).

/PREVIOUS*[=n]* — (CLI32 only.) Same as /P.

DIRECTORY Example 1

The following sequence of commands moves around user JAN's directory tree.

```
) DIRECTORY ↵                    (Displays the working directory.)
:UDD:JAN:ALPHA:BETA

) DIRECTORY/I ↵                  (Moves to the initial directory.)
) DIRECTORY ↵
:UDD:JAN

) DIRECTORY  ALPHA ↵             (Moves to the subdirectory ALPHA.)
) DIRECTORY ↵
:UDD:JAN:ALPHA

) DIRECTORY  ^GAMMA ↵            (Moves up to JAN and then down to GAMMA,
                                  using the prefix ^.)
) DIRECTORY ↵
:UDD:JAN:GAMMA

) DIRECTORY/I DELTA ↵            (Moves to :UDD:JAN:DELTA, using a pathname
                                  relative to the initial directory.)
) DIRECTORY ↵
:UDD:JAN:DELTA
```

DIRECTORY Example 2

```
) XEQ  MYPROGRAM ↵
Error:  File does not exist
XEQ,MYPROGRAM

) DIRECTORY : UDD:TOM:PROGRAMS ↵
) FILESTATUS  MYPROGRAM.PR ↵
MYPROGRAM.PR

) XEQ MYPROGRAM ↵
...
```

The first command tries to execute MYPROGRAM, but the system cannot find the program. (The specified program does not reside in either the working directory or a directory on the current search list.) The DIRECTORY command changes the working directory, and the FILESTATUS command verifies that the program file resides there. The XEQ command now succeeds.

Note that if user TOM wanted to run this program without changing the working directory, TOM could either provide a full pathname to the XEQ command via

```
) XEQ :UDD:TOM:PROGRAMS:MYPROGRAM ↵
```

or else add :UDD:TOM:PROGRAMS to the current search list via

```
) SEARCHLIST [!SEARCHLIST], :UDD:TOM:PROGRAMS ↵
```

and then type

```
) XEQ MYPROGRAM ↵
```

!DIRECTORY *Pseudomacro*

Expands to the full pathname of the working directory.

Format

[!DIRECTORY]

This pseudomacro represents the full pathname (beginning at the root directory) of the working directory. (A *pathname* argument applies only if you are using the /I switch, described later.)

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

Why Use It?

Use the !DIRECTORY pseudomacro to include the full pathname of the working directory in an argument to a CLI command or pseudomacro.

Macro Name Switches

/I	Expands to the initial working directory.
/INITIAL	(CLI32 only). Same as /I.
/I pathname	Expands to the directory specified by pathname, which starts from the initial working directory. In other words, pathname is relative to the initial working directory. See the last part of Example 2. The pathname must specify a valid (existing) directory. If you specify a filename rather than a directory name (a file of a type other than CPD, DIR, or LDU), the pseudomacro will not work.
/INITIAL pathname	(CLI32 only). Same as /I pathname.
/LEVEL=n	(CLI32 only.) Expands to the working directory 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.

!DIRECTORY (continued)

/P*[=n]*	Without =*n*, sets the working directory to the previous environment's working directory. (Omit the pathname argument.) With =*n* (CLI32 only), expands to the working directory 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.

!DIRECTORY Example 1

(within a macro)

```
write The termination message is [!string].
write The working directory is [!directory].
write The current search list is [!searchlist].
```

Upon termination of a program, this macro reports the termination message (which was directed to the CLI String), the working directory, and the current search list. This information could help you determine the cause of an unexpected termination.

!DIRECTORY Example 2

In the following example, we first use the !DIRECTORY pseudomacro to type the pathname of the current working directory. We then append /I to write the pathname of the initial directory.

```
) WRITE The working directory is [!DIRECTORY]. ↵
The working directory is :UTIL.

) WRITE The initial directory is [!DIRECTORY/I]. ↵
The initial directory is :UDD:LEE.
```

Next, we append the /I pathname switch to write the pathname (starting at the initial directory) of the example directory TEST.

```
) WRITE [!DIRECTORY/I TEST] ↵
:UDD:LEE:TEST
```

DISCONNECT *Command*

Breaks a connection with the specified server process.

Format

DISCONNECT process-ID

This command directs the system to break the customer–server connection between the specified server process and the CLI process.

A *server* process performs tasks on behalf of other processes. A process that wants to use a server process (called a *customer* of that server) must establish a connection with the server before requesting service.

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

Why Use It?

Use the DISCONNECT command to break the customer–server connection when you no longer need to use the server process, or when the server has terminated. Normally there is a limit to the number of customer–server connections allowed on a system, so you should use this command to break a connection that is no longer needed.

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=.

DISCONNECT Example

```
) CONNECT OP:SERVER ↵
Server's PID:   14
...
...
) DISCONNECT 14 ↵
```

The first command establishes a connection with the server process OP:SERVER. The system reports that server's PID. Later, the DISCONNECT command uses that PID number to break the connection.

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.

DISPLAY (continued)

/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.

DUMP (continued)

/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.

DUMP_II (continued)

/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.

DUMP_II (continued)

/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.

DUMP_II (continued)

/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.

DUMP_II Example 4

```
) DUMP_II/V/AFTER/TLM=23-MAY-90 @MTB0:1 -.F77 -.SR ↵
```

... (Utility displays list of files dumped) ...

This command dumps all FORTRAN 77 and assembly language source files that were last modified (or created) after May 23, 1990, from the working directory. It dumps them to to the second tape file of the tape on unit 0. It verifies this by displaying the names of the dumped files.

DUMP_II Example 5

```
) SUPERUSER ON ↵
Su) MOUNT/VOLID=V1/VOLID=V2/VOLID=V3/EXTEND XTAPE Time for backup. ↵
```

(The system operator mounts the first tape.)

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

(The system dump occurs to multiple volumes.)

```
Su) DISMOUNT XTAPE 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_II command then dumps all files in the system, maintaining directory structure, to tape file FILESET1 using the link name XTAPE (that was created in the initial user directory). The dump can use the three tape volumes specified with the /VOLID switches; the program will prompt for others if needed, as directed by the /EXTEND switch. The dump listing goes to file SYSTEM_BACKUP in the initial user directory. After the dump is done, the person types DISMOUNT, prompting the operator to dismount the last tape.

DUMP_II Example 6

```
) DUMP_II/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. Those files will be loaded with the default ACLs.

!EDIRECTORY *Pseudomacro*

Expands to the directory portion of a pathname.

Format

[!EDIRECTORY pathname *[...]*]

This pseudomacro represents the directory portion of the specified pathname(s).

The directory portion of a pathname is actually the pathname of the parent directory. If the argument is a simple filename, the pseudomacro returns a null value. In the following sample pathnames, the directory portion is underlined.

:UDD:TOM:REPORTS:FEB_1990
:UDD:TOM:REPORTS
@CON6
MYFILE (simple filename; no directory portion exists)

- No templates.
- No argument switches.
- No macro name switches.
- Requirement: *Standard.*
- See also: !EEXTENSION, !EFILENAME, !ENAME, !EPREFIX, Figure 2–3.

Why Use It?

Use the !EDIRECTORY pseudomacro to extract the directory portion of a pathname, and use that information as an argument to a CLI command or macro. Example 1 shows how a macro can use !EDIRECTORY to move to the directory that contains a file whose pathname was passed to the macro as an argument.

!EDIRECTORY Example 1

```
directory [!edirectory %1%]
```

This macro statement extracts the directory portion of the pathname passed as an argument to the macro, and then makes that directory the working directory.

!EDIRECTORY Example 2

Another macro contains the following two lines:

```
permanence %1% on
permanence [!edirectory [!pathname %1%]] on
```

The first statement turns on permanence for the file specified in the pathname argument passed to the macro. The second statement turns on permanence for the directory that contains the file. (The !PATHNAME pseudomacro is used in case the argument is a simple filename without directory information.)

!EDIRECTORY Example 3

```
) WRITE [!EDIRECTORY :UDD:JR:ENG:MATH.PR] )
:UDD:JR:ENG
```

This command shows the directory portion of a pathname.

!EEXTENSION *Pseudomacro*

Expands to the filename suffix portion of a pathname.

Format

[!EEXTENSION pathname *[...]*]

This pseudomacro represents the filename suffix portion of the specified pathname(s).

The filename suffix begins with the last period following the rightmost colon or prefix character, and continues through the end of the pathname string. (If there is no period following the rightmost colon or prefix, the filename contains no suffix, and the pseudomacro returns a null.) In the following pathname samples, the suffix portion is underlined.

:UDD:TOM:PROGRAMS.DIR:MORTGAGE.SRC
:UDD:TOM:PROGRAMS.DIR
:UDD:TOM:PROGRAMS:MYPROG.PR
:UDD:TOM(no filename suffix)

- No templates.
- No argument switches.
- No macro name switches.
- Requirement: *Standard.*
- See also: !EDIRECTORY, !EFILENAME, !ENAME, !EPREFIX, Figure 2–3.

Why Use It?

Use the !EEXTENSION pseudomacro to extract the filename suffix from a pathname, and use that suffix as an argument to a CLI command or macro. You can use this pseudomacro to check for a specific suffix (see Example 1), or to assign the suffix of one file to another (see Example 2).

!EEXTENSION Example 1

The macro SHOW.CLI executes the CLI TYPE command, but first checks to ensure that you do not try to display the contents of a program file. The macro contains the following lines

```
comment This is macro SHOW.CLI.
[!nequal, [!eextension %1%], .pr]
        type %1%
[!else]
        write This is a program file — you cannot read it.
[!end]
```

The first line extracts the filename suffix from the pathname argument that is passed to the macro. If the suffix is not .PR (by convention, a program file), the macro displays the file on the screen. If the file from the pathname argument is a program file, as is :UDD:ALICE:WORK5:PROGRAM_03.PR, the macro displays a message telling the user that the file cannot be displayed.

!EEXTENSION Example 2

```
create temp[!eextension %1%]
```

This macro command creates a temporary file (TEMP), and appends the same filename suffix used by the first argument passed to the macro.

If the argument is DATA22.IN, the command creates TEMP.IN; or if the argument is REPORT.JUL89, the macro creates TEMP.JUL89. If the argument has no suffix, the macro creates TEMP.

!EEXTENSION Example 3

```
) WRITE [!EEXTENSION :UTIL:BUILD.PR :UDD:LEE:REPORT.TXT] ↵
.PR .TXT
```

This command shows the extension portion of the pathnames :UTIL:BUILD.PR and :UDD:LEE:REPORT.TXT.

!EFILENAME *Pseudomacro*

Expands to the filename portion of a pathname.

Format

[!EFILENAME pathname *[...]*]

This pseudomacro represents the filename portion of the file(s) specified by pathname.

The filename portion of a pathname is the name of the file itself, including any filename suffix. It begins immediately after the rightmost colon or prefix and includes all characters to the right. In the following sample pathnames, the filename portion is underlined.

:UDD:TOM:REPORTS:FEB_1990
:UDD:TOM:PROGRAMS:MORTGAGE.PR
:UTIL:SORT_3.30:SORT.CLI
MYFILE

- No templates.
- No argument switches.
- No macro name switches.
- Requirement: *Standard.*
- See also: !EDIRECTORY, !EEXTENSION, !ENAME, !EPREFIX, Figure 2–3.

Why Use It?

Use the !EFILENAME pseudomacro to extract the filename portion of a pathname, and use that information as an argument to a CLI command or macro. This pseudomacro lets you strip parent directory information from a pathname.

!EFILENAME Example 1

) WRITE [!EFILENAME [!LISTFILE]] ↵

This command displays the filename of the current list file without the complete pathname.

!EFILENAME Example 2

(within a macro)

```
[!nequal, [!efilename temp], %1%]
        xeq  myprogram %1%
[!else]
        write This program uses the filename TEMP for temporary data.
        write Please use a different name for your file.
[!end]
```

This macro checks the filename argument passed to it. If the filename is not TEMP, the macro executes a program called MYPROGRAM. But if the filename argument is TEMP, the macro warns the user to use a different name for the file.

!EFILENAME Example 3

```
) WRITE [!EFILENAME :MACROS:SHIFT.CLI :UDD:LEE:MYFILE.PR] ↵
SHIFT.CLI  MYFILE.PR
```

This command shows the filename portion of the full file pathnames :MACROS:SHIFT.CLI and :UDD:LEE:MYFILE.PR.

!ELSE *Pseudomacro*

Begins a sequence of statements that will execute when the result of a conditional test is false.

Format

```
conditional pseudomacro
        (commands executed if condition is true)
        ...
[!ELSE]
        (commands executed if condition is false)
        ...
[!END]
```

This pseudomacro begins a series of statements to be executed if the result of a conditional pseudomacro (!EQUAL, !NEQUAL, !UEQ, !UNE, !ULE, !ULT, !UGE, or !UGT) proves false. The CLI then executes the command(s) that follow !ELSE until the corresponding !END statement. If the condition is true, the statements up to !ELSE are executed, and those that follow are ignored.

- No templates.
- No macro name switches.
- Requirement: *Standard.*
- See also: !EQUAL, !NEQUAL, !UEQ, !UNE, !UGT, !UGE, !ULT, !ULE (conditional pseudomacros), and !END.

Why Use It?

Use the !ELSE pseudomacro to define a sequence of one or more statements to be executed when a condition tests false. Chapter 4 explains using pseudomacros.

!ELSE Example

(Used within a macro named PROGRAM.CLI)

```
[!equal, %1%, ]
        WRITE You did not supply an argument.
[!else]
        XEQ MYPROGRAM %1%
[!end]
```

Macro PROGRAM.CLI ensures that the user has supplied a necessary argument in the command line before the macro executes MYPROGRAM. If the argument is missing (that is, null), the macro displays a message. The !ELSE pseudomacro begins an alternative series of steps to be executed if the !EQUAL statement proves false (that is, the argument is *not* null). Examples of command lines that result in the execution of the WRITE and XEQ statements are, respectively,

```
) PROGRAM ↵
```

and

```
) PROGRAM :UDD3:LISA:TESTFILE_52.OUT ↵
```

!ENAME *Pseudomacro*

Expands to the name portion of a pathname.

Format

[!ENAME pathname *[...]*]

This pseudomacro represents the name portion of the specified pathname(s).

The name portion is the simple filename excluding any filename suffix. It begins immediately following the rightmost prefix, if any, and ends immediately before the rightmost period, if any. (If there is no prefix character in the pathname, the name portion begins with the leftmost character; if there is no period in the string, the name ends with the rightmost character.) The prefix characters are :, @, =, and ^. In the following sample pathnames, the name portion is underlined.

:UDD:SALLY:PROGRAMS:MORTGAGE.F77
:UDD:SALLY:SOURCE.PL1
:UDD:SALLY:SOURCE.PL1.BU
:UTIL:CONVERT.PR
@CON24
MYFILE

- No templates.
- No argument switches.
- No macro name switches.
- Requirement: *Standard.*
- See also: !EDIRECTORY, !EEXTENSION, !EFILENAME, !EPREFIX, Figure 2–3.

Why Use It?

Use the !ENAME pseudomacro to extract the name portion of a pathname, and use that name as an argument to a CLI command or macro.

!ENAME Example 1

(within a macro)

```
.
delete/2=ignore [!ename %1%].ls
listfile [!ename %1%].ls.
```

This macro accepts a program file pathname as an argument. Before executing the program, the macro deletes any file with the same name as the program file (plus the .ls suffix); then it sets the list file to a filename of the form it just deleted.

!ENAME Example 2

```
) WRITE [!ENAME :UDD:JR:REPORT.TXT] ↵
REPORT
```

This command shows the name portion of the full document pathname :UDD:JR:REPORT.TXT.

!ENAME Example 3

```
) WRITE [!ENAME ^MYFILE.TXT.ED] ↵
MYFILE.TXT
```

This command shows the name portion of the pathname MYFILE.TXT.ED.

!ENAME Example 4

```
) WRITE [!ENAME :UDD:LEE:DOC1.TXT] ↵
DOC1
```

```
) WRITE [!ENAME :UDD:SANDY:TEST.RESULTS.WRD] ↵
TEST.RESULTS
```

These commands show the name portions of the document pathnames :UDD:LEE:DOC1.TXT and :UDD:SANDY:TEST.RESULTS.WRD.

!END *Pseudomacro*

Ends a series of statements that begins with a conditional pseudomacro.

Format

conditional pseudomacro

...

[!ELSE]

...

...

[!END]

This pseudomacro marks the end of a series of statements that begins with a conditional pseudomacro (!EQUAL, !NEQUAL, !UEQ, !UNE, !ULE, !ULT, !UGE, or !UGT).

- No templates.
- No argument switches.
- No macro name switches.
- Requirement: *Standard.*
- See also: !EQUAL, !NEQUAL, !ELSE, !UEQ, !UNE, !UGT, !UGE, !ULT, !ULE.

Why Use It?

Use the !END pseudomacro to close a sequence of statements that begins with a conditional pseudomacro.

If you execute a macro that does not have an !END statement for each conditional pseudomacro, the CLI displays one of the following prompts:

/)	means you are in a true code path
\)	means you are in a false code path

You can enter any CLI command following such a prompt. To close the code path and have the macro execute the commands you entered, type [!END] following the prompt. Or you can abort the macro by issuing a console interrupt (CTRL–C CTRL–A).

For more on conditional pseudomacros, see Chapter 4.

!END Example 1

(Used within a macro named PROGRAM.CLI)

```
[!equal, %1%, ]
        WRITE You did not supply an argument.
[!else]
        XEQ  MYPROGRAM %1%
[!end]
```

Macro PROGRAM.CLI ensures that the user has supplied a necessary argument in the command line before the macro executes MYPROGRAM. If the argument is missing (that is, null), the macro displays a message. The !END pseudomacro marks the end of the series of statements that begins with the !EQUAL pseudomacro.

!END Example 2

```
[!equal, [!operator], on]
        WRITE The system operator is on duty.
[!end]
```

This macro simply tests to see if the system operator is on duty. If so, it displays a message to this effect. Otherwise, it does nothing. Note that an [!ELSE] pseudomacro does not have to appear before [!END].

!END Example 3

```
) [!EQUAL 1,1]WRITE Argument1 equals argument2 [!END] ↵
Argument1 equals argument2
```

This example shows how [!END] terminates a conditional that begins on the same line. There is no space immediately before the WRITE command. Within a macro a more readable structure is

```
[!EQUAL 1,1]
        WRITE Argument 1 equals argument 2
[!END]
```

!END Example 4

```
) [!EQUAL,1,1]WRITE EQUAL ↵
!) [!END] ↵
EQUAL
```

This example shows what happens when you have a condition that isn't terminated by [!END]. (Notice the exclamation mark that appears before the CLI prompt. This indicates that you're missing an [!END], and that the condition is true.) You must type [!END] to execute the command line.

!EPREFIX *Pseudomacro*

Expands to the prefix portion of a pathname.

Format

[!EPREFIX pathname *[...]*]

This pseudomacro represents the prefix portion of the file(s) specified by pathname.

The prefix portion of a filename begins with the leftmost character and continues up to the rightmost prefix character (:, @, =, ^). If there is no prefix character (that is, the pathname is a simple filename), the pseudomacro returns a null value. In the following sample pathnames, the prefix portion is underlined.

:UDD:JOAN:PROGRAMS:MORTGAGE.SRC
JOAN:PROGRAMS.DIR:MORTGAGE.F77
@CON6
MYFILE (no prefix)

- No templates.
- No argument switches.
- No macroname switches.
- Requirement: *Standard.*
- See also: !EDIRECTORY, !EEXTENSION, !EFILENAME, !ENAME, Figure 2–3.

Why Use It?

Use the !EPREFIX pseudomacro to extract the prefix portion of a pathname, and use the prefix as an argument to a CLI command or macro. One common use is to extract the prefix from one pathname, and then append a simple filename to that prefix.

!EPREFIX Example 1

(Used within a macro)

```
[!nequal, [!eprefix %1%], :UDD]
        write The file must reside in your initial directory – :UDD:[!USERNAME]
[!else]
        ...
[!end]
```

This macro checks the prefix of the pathname that was passed to the macro as an argument. If the prefix is not :UDD (the user directory directory), the macro displays a message telling the user that the file must reside in the user's initial directory.

!EPREFIX Example 2

```
CREATE [!EPREFIX [!PATHNAME %1%]] TEMP
```

This macro statement creates a file called TEMP in the same directory as the file that was passed to the macro as an argument. The !PATHNAME pseudomacro is used in case the argument is a simple pathname without a prefix.

!EPREFIX Example 3

```
) WRITE [!EPREFIX :UDD:SANDY:STATS  :UDD:LEE:TEST:MYFILE] ↵
:UDD:SANDY: :UDD:LEE:TEST:
```

This command shows the prefix portions of the pathnames :UDD:SANDY:STATS and :UDD:LEE:TEST:MYFILE.

!EQUAL *Pseudomacro*

Tests two values for equality, and continues execution based on the result.

Format

[!EQUAL, argument1, argument2]

...

...

[[!ELSE]]

...

...

[!END]

This pseudomacro compares the two arguments character by character; the CLI considers uppercase and lowercase letters as equal.

If the arguments are equal (the condition is true), then:

> the CLI executes the statements bounded by the corresponding !END statement. If !ELSE appears, however, the CLI executes only the statements that precede it, and ignores the statements that follow !ELSE (up to !END).

If the arguments are unequal (the condition is false), then:

> the CLI ignores the statements bounded by the corresponding !END statement, unless !ELSE appears. In that case, the CLI ignores the statements that precede !ELSE, but executes those that follow !ELSE (up to the corresponding !END statement).

Note that the !EQUAL pseudomacro compares the arguments as strings. This means that when [!EQUAL] compares 019 and 19, it will produce a false condition. To compare arguments numerically, you must use the !UEQ pseudomacro.

- No templates.
- No argument switches.
- No macro name switches.
- Requirement: *Standard.*
- See also: !UEQ, !END, !ELSE, and !NEQUAL.

!EQUAL (continued)

Why Use It?

Use the !EQUAL pseudomacro to execute one or more commands only if two values are equal. You can include the !ELSE pseudomacro to take other action if the two values are unequal. !EQUAL is best used for strings; if you want to compare integer values, use !UEQ instead.

A macro can use !EQUAL to display a message if an argument is null. (See Example 1.) Or a macro can perform an action depending on a value entered at the terminal. (See Example 2.)

If you are comparing text strings that contain any separator characters, enclose each argument in parentheses. (See Example 3.)

Using conditional pseudomacros is explained in Chapter 4.

!EQUAL Example 1

This macro checks to see if the argument passed to it is null.

```
[!equal, %1%, ]
        write You must supply an argument with this macro.
[!else]
        ...
        ...
[!end]
```

If the person who executes the macro does not specify an argument in the command line, the macro displays an error message. Otherwise, the macro executes the statements following the !ELSE pseudomacro (which use the argument information).

!EQUAL Example 2

The first statement in the following macro requests a response and then compares it with the letter Y. If the two are equal, the macro issues the QPRINT command; otherwise, the macro resumes processing following the !END statement.

```
[!equal, [!read Do you want to print file %1%?,,],Y]
        qprint %1%
[!end]
```

The next version of the same macro includes an !ELSE pseudomacro to display a message if the response is not Y.

```
[!equal, [!read Do you want to print file %1%?,,],Y]
        qprint %1%
[!else]
        write File %1% not printed.
[!end]
```

!EQUAL Example 3

[!equal, ([!string]),(END OF FILE)]

In this macro statement, the !EQUAL pseudomacro compares string arguments that include separator characters. You must enclose both arguments in parentheses.

!EQUAL Example 4

) STRING HELLO↓
) [!EQUAL [!STRING],HELLO]WRITE EQUAL [!ELSE]WRITE NOT EQUAL [!END] ↓
EQUAL

) [!EQUAL [!STRING],BYE]WRITE EQUAL [!ELSE]WRITE NOT EQUAL [!END] ↓
NOT EQUAL

For both commands, the CLI String equals HELLO. Therefore, in the first command line the !EQUAL pseudomacro is true. In this case the CLI executes the command between [!EQUAL] and [!ELSE], and ignores the command between [!ELSE] and [!END].

In the second command line the !EQUAL pseudomacro is false. In this case the CLI executes the command between [!ELSE] and [!END], and ignores the command between [!EQUAL] and [!ELSE].

In both command lines there is no space immediately before the WRITE commands.

!EQUAL Example 5

In this example, the !EQUAL pseudomacro is used to create a false condition (that 1 equals 2) so that the CLI will ignore everything up to the !END statement. This allows you to enter a comment into a macro without the restrictions that come with the CLI COMMENT command.

[!equal, 1, 2]
The preceding pseudomacro will always be false, so the CLI will not try to execute any of these comment statements. If we used the CLI COMMENT command, we would have trouble using special characters such as a comma, which the CLI would interpret as a delimiter, or parentheses and angle brackets (which the CLI would try to expand), or even a semicolon; a semicolon would terminate the COMMENT command altogether.
[!end]

EXECUTE *Command*

Executes a program.

Format

EXECUTE pathname*[.PR] [argument] [...]*

The command executes the program (specified by pathname) as a son (subordinate) swappable process of the CLI. The CLI first looks for pathname.PR; if that does not exist, the CLI looks for pathname. The pathname must be an executable program.

If the program requires or accepts arguments, you can specify them. For information about how a program started from the CLI can gain access to the arguments and switches included in the CLI command line, see the ?GTMES system call in the *AOS/VS, AOS/VS II, and AOS/RT32 System Call Dictionary, ?A through ?Q.*

The new process has the same priority and privileges as its parent CLI process. It also uses the same generic files, with the possible exception of @LIST and @DATA; the new program uses the current list file and data file settings, which may be different from the CLI's.

The CLI blocks until the program terminates. If the program returns an error condition, the CLI tries to interpret the error code.

EXECUTE works the same way as XEQ or X.

- No templates.
- Use any argument switches appropriate for the program.
- Requirement: *Standard.*
- See also: PROCESS (to specify detailed process information about the program) and CHAIN (to overwrite the CLI with a new program).

Program Input

The program's use of @INPUT depends on whether you execute the program in batch mode or not. In batch mode, you can use the /I or /M switch to provide input. If you are working at a terminal, you can enter input through the terminal.

Why Use It?

Use the EXECUTE (or XEQ or X) command to run a program, and then return to the CLI when the program terminates.

NOTE: You do not use this command to run a macro. To execute a macro, simply type the macro name.

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

!EXPLODE Example 2

The macro TODAY.CLI uses the !EXPLODE pseudomacro and the system date to display a message such as

Today is November 24, 1990.

To call TODAY.CLI from your startup macro (to display the date each time you log on), you would include the command TODAY/

TODAY.CLI interprets individual characters of the system date to determine the month, and uses the digits for the day and year to build the message. (The macro calls itself and supplies the system date in exploded form.) The macro follows.

```
comment  This is macro TODAY.CLI.  The macro converts the
comment  system date to a more readable form.
comment  For example, if the argument is "2 4 - N O V - 9 0" the
comment  macro displays "Today is November 24, 1990."
comment  The argument to this macro is the current date exploded.
comment  It may come from the calling command or within the macro.
[!nequal,%1%,]
    comment Argument 1 was the date. Expand the month abbreviation.
    push; prompt pop
    [!equal %4% F]
        string February
    [!else]
        [!equal %4% S]
        string September
        [!else]
            [!equal %4% O]
            string October
            [!else]
                [!equal %4% N]
                string November
                [!else]
                    [!equal %4% D]
                    string December
                    [!else]
                        [!equal %4% M]
                            [!equal %6% R]
                            string March
                            [!else]
                            string May
                            [!end]
                            [!else]
                            [!equal %4% A]
                                [!equal %5% P]
                                    string April
                                [!else]
                                    string August
                                [!end]
```

```
comment String is not a month whose name begins with A. Test Js.
                                [!else]
                                [!equal %4% J]
                                    [!equal %5% A]
                                    string January
                                        [!else]
                                        [!equal %6% N]
                                        string June
                                        [!else]
                                        string July
                                        [!end]
                                    [!end]
                                [!end]
                            [!end]
                        [!end]
                    [!end]
                [!end]
            [!end]
        [!end]
    [!end]
    write Today is [!string] %1%%2%[!ascii 254] 19%9%%0%.
    pop
[!else]
comment  Argument 1 was omitted.  Call the macro with the current
comment  date exploded.
    %0% [!explode [!date]]
[!end]
```

Next are five executions of macro TODAY.CLI.

) TODAY ↵
Today is November 24, 1991.

) TODAY 2 3 – N O V – 9 1 ↵
Today is November 23, 1991.

) TODAY 3 – N O V – 9 1 ↵
Today is October 3–, 1991.
(All days–of–the–month must contain two digits.)

) TODAY 0 3 – N O V – 9 1 ↵
Today is November 03, 1991.

) TODAY 03–NOV–91 ↵
Error: Pseudomacro has wrong number of arguments
Level 1
[!equal,F]

!EXPLODE Example 3

In this example suppose that you have a macro called Z.CLI, which takes an argument and writes it five times via the following command.

```
WRITE %1% %1% %1% %1% %1%
```

Therefore, typing [!EXPLODE ZX] would result in the following.

```
) [!EXPLODE ZX] ↵
XXXXX
```

By exploding ZX into Z,X you have instructed the system to execute the macro Z.CLI and to use X as this macro's argument.

FCU *Utility*

Sets nonstandard form parameters for files to be printed.

Format

XEQ FCU

The Forms Control Utility (FCU) lets you specify horizontal tabs and vertical forms settings for a user text file or a forms file in directory :UTIL:FORMS. A file's User Data Area (UDA) contains forms specifications for the file, even if the file is empty. The specifications do not affect the file length as shown in a FILESTATUS display. You can discover whether a file has a UDA with the FILESTATUS/UDA command.

New specifications you can set with FCU include characters per line, lines per page, and tab stops. Also, you can specify a total of 12 channels (lines on the form) for printing. Channel control will work only if the text file will include appropriate control characters and be printed on a printer with vertical format control (data channel line printer).

A file with forms specifications in its UDA can serve for the printing of other files via the a command of the form

QPRINT/FORMS=form–filename text–filename

Or the file can serve for the printing of text *within itself* via a command of the form

QPRINT text–filename

Requirements for the QPRINT/FORMS= command are more formal than those for the QPRINT command, as follows. All forms files specified with

QPRINT/FORMS=form–filename

must be in directory :UTIL:FORMS. So if you want form specs to be available for printing text in other files, you must move the forms file to directory :UTIL:FORMS after running FCU on it. After anyone types a QPRINT/FORMS= command, the system holds the print job until the system operator issues an EXEC FORMS command to tell EXEC that the correct form has been inserted in the printer.

Printing with QPRINT without the /FORMS= switch is much less formal. The file with text and forms specs can be in any directory, and the system will print it immediately, using the forms specs in its UDA, without waiting for the operator to notify EXEC.

- Does not accept templates.
- Accepts a utility switch (described later).
- Requirement: *Standard* (Execute access to the program file in :UTIL).
- See also: QPRINT.

FCU (continued)

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).

FCU (continued)

7. *Bottom of Form (Channel 12) Line Number (top–lastline) [lastline] ?*

   Type the number of the last line on which you want to print. This number also becomes the setting for channel 12 of the VFU tape. The default value (shown in square brackets) is the line number you specified for the *Form length* query. The top line and last line you specified are shown in parentheses.

8. *VFU Tape (Line numbers top–lastline, Channels 2–11, OR STANDARD)*
   []
   ?

   Specify a VFU tape only if the printer you will use has one.

   To specify no VFU tape, type STANDARD (STA) or press NEW LINE. To specify a channel and line number, type the line number you want to advance to, followed by the channel number. (For example, 10–2 signifies that the printer will advance to line 10 when it encounters the code for channel 2. Table 5–2 lists the channel codes.) You can specify more than one line number per channel number. In this case, when it encounters the code for the channel, the line printer advances to the next line number that you have associated with this channel.

9. *Output to Pathname*
   [pathname] ?

   If you want to copy the forms control specifications to another file, type that file's pathname. To write the specifications to the default pathname, press NEW LINE only.

   The *Command?* prompt reappears. If you are finished using FCU, type B; otherwise enter the appropriate command.

FCU Switches

/L=pathname	Use this switch if you want to list the forms control specifications for a file (L command).

Octal Codes and Their Printer Responses

In response to question 8 of the FCU dialog, you can assign values to each of the channels in Table 5–2 (except channels 1 and 12, which are reserved for top of form and bottom of form respectively.) When it encounters the 2–byte octal code for a channel in the file it is printing, the line printer will quickly advance to the line you specified in question 8. For example, suppose you have matched channel 2 with line 10 of the form in question 8 and placed the 2–byte octal code <022><101> (for channel 2) in the file you are about to print. When the printer encounters these two bytes it will quickly advance the paper to line 10.

Use the octal codes as shown in Table 5–8; do not set the parity bit is not set for the codes. You can use the DISPLAY utility to make sure that you have the correct octal codes in your text.

Table 5–8 Vertical Forms Unit Channel Codes

Channel	Octal Codes	ASCII Codes
1	<022> <100>	CTRL–R @
2	<022> <101>	CTRL–R A
3	<022> <102>	CTRL–R B
4	<022> <103>	CTRL–R C
5	<022> <104>	CTRL–R D
6	<022> <105>	CTRL–R E
7	<022> <106>	CTRL–R F
8	<022> <107>	CTRL–R G
9	<022> <110>	CTRL–R H
10	<022> <111>	CTRL–R I
11	<022> <112>	CTRL–R J
12	<022> <113>	CTRL–R K

Table 5–9 shows the effect of other ASCII codes on the VFU of the line printer. These codes cause the line printer to advance a certain number of lines relative to the current position. You cannot change these settings with the FCU.

When it encounters the ASCII codes in your text, the line printer automatically advances the number of lines specified in the Step Count column. For example, suppose you have placed the 2–byte code <022><135> in your text file. When the line printer encounters this code it will advance the paper 13 lines from wherever it is.

Note that in the octal codes, the parity bit is not set. Use the DISPLAY utility to make sure you have the correct code in your text.

Table 5–9 Vertical Forms Unit Step Count Codes

Step Count	Octal Codes	ASCII Codes
0	<022> <120>	CTRL–R P
1	<022> <121>	CTRL–R Q
2	<022> <122>	CTRL–R R
3	<022> <123>	CTRL–R S
4	<022> <124>	CTRL–R T
5	<022> <125>	CTRL–R U
6	<022> <126>	CTRL–R V
7	<022> <127>	CTRL–R W
8	<022> <130>	CTRL–R X
9	<022> <131>	CTRL–R Y
10	<022> <132>	CTRL–R Z
11	<022> <133>	CTRL–R [
12	<022> <134>	CTRL–R \
13	<022> <135>	CTRL–R]
14	<022> <136>	CTRL–R ^
15	<022> <137>	CTRL–R –

FCU Example

) XEQ FCU ↵

AOS/VS Forms Control Utility Revision n.nn 27-NOV-89 13:00:00
Type 'Help' for instructions

Command? C ↵ (Create command creates forms specifications.)
Pathname? MY_FORMS_FILE ↵

Characters per line (16–255)
[80]? ↵

Tab stops (2–79, OR STANDARD)
[8,16,24,32,40,48,56,64,72]
? 10 ↵
? 20 ↵
? 30 ↵
? 40 ↵
? 50 ↵
? ↵

Form length in Lines Per Page (6–144)
[66] ? 60 ↵

Top of Form (Channel 1) Line Number (1–60)
[4] ? ↵

Bottom of Form (Channel 12) Line Number (4–60)
[60] ? ↵

VFU Tape (Line numbers (4–60), Channels 2–11, OR STANDARD)
[]
? ↵

Output to Pathname
[:UDD:ZONIS:MY_FORMS_FILE] ? ↵ (Writes the new forms specifications to the UDA of file MY_FORMS_FILE.)

FCU Example (continued)

Command? T↵ (Types the specifications on the terminal.)
Pathname? FILE1 ↵ (Specify the file whose specs to describe.)

Pathname
[:UDD:ZONIS:FILE1]
Characters per line
[80]
Tab stops
[10,20,30,40,50]
Form length in Lines Per Page
[60]
Top of Form (Channel 1) Line Number
[4]
Bottom of Form (Channel 12) Line Number
[60]
VFU Tape
[]

Command? B ↵ (Exits from FCU.)

FCU terminating 27–NOV–90 13:04:46

In this example, user ZONIS created format specifications for FILE1. He chose the default of 80 characters per line, but did not choose the default tabs. Instead, he placed tabs at 10, 20, 30, 40, and 50. He chose a form length of 60 lines per page, with printing to begin on line 4 and end on line 60 (default values). By pressing NEW LINE in response to the VFU query, he specified a standard (null) VFU tape. All output goes to the User Data Area of file FILE1. Finally, user ZONIS typed out the forms control specifications for FILE1 to check their accuracy before leaving the FCU program.

FILCOM *Utility*

Compares two files.

Format

XEQ FILCOM pathname1 pathname2

The FILCOM utility compares two files, 16–bit word by 16–bit word, and displays the address and contents of words that differ. FILCOM also describes the total number of differing words in the files and their User Data Areas (UDAs).

FILCOM displays addresses and word contents in octal, unless you specify otherwise with the /RADIX switch. It omits leading zeros from the display. It shows the last byte in a file as < *n*>——.

- Does not accept templates.
- Accepts utility switches (described later).
- Requirement: *Standard* (Execute access to the program file in :UTIL).
- See also: DISPLAY, SCOM, BROWSE.

Why Use It?

FILCOM is designed for use in situations where you care about the numeric address and contents of differing parts of the files, as with binary files like program files. You can access the addresses that FILCOM displays via a disk file editor utility (FED).

FILCOM can also tell you if file User Data Areas (UDAs) differ, or if one file has a UDA and the other does not. It will display *filename does not have a UDA* if one file has a UDA and the other does not. FILCOM does not display the contents of UDAs; to display UDA contents, use the DISPLAY utility with the /LISTUDA switch.

For a line–by–line comparison of text files, use the SCOM utility instead of FILCOM.

FILCOM Switches

/ADR_RADIX=n	Uses the specified radix when displaying the file address; n can be from 2 through 16. The default radix is 8.
/HEADER	Displays the pathnames of the files being compared. (Header information may change, so do not depend on any specific format or content.)
/L	Writes output to @LIST instead of to @OUTPUT.

FILCOM (continued)

/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.

!FILENAMES (continued)

/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.

FILESTATUS (continued)

/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.

FILESTATUS (continued)

/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.

GROUPLIST *Command*

Sets or displays a group list (CLI32 with AOS/VS II only).

Format

GROUPLIST *[groupname] [...]*

This command allows you to set or display your group list. If you omit arguments, the CLI displays your current group list. To join one or more groups, use the GROUPLIST command with the group name(s) as argument(s). A group list can include up to eight group names.

A user group is a set of users, associated by group name. You can join a group only if your username is included in a file named for the group within directory :GROUPS. If you try to join a group that doesn't include your username, the system will display the error message *User cannot be in group*. If the group does not exist, the CLI will display *Group does not exist*. (If the :GROUPS directory doesn't exist, the system will display *:GROUPS directory does not exist;* in this case, you may want to contact the system manager.

All groups that a user has joined are kept in a group list, which is analogous to a search list. At logon, your group list is empty; it contains no group names. You can learn the names of available groups from the system manager. Or you have read access to :GROUPS, you can learn the names of all groups by typing

```
) FILESTATUS :GROUPS:+ ↵
```

The group list, like the search list, is part of the CLI environment; you can specify a different group list on each level. A pseudomacro, !GROUPLIST, displays the current group list.

- No templates.
- No argument switches.
- Requirement: *Standard* (your username must be included in the group file in directory :GROUPS).
- See also: ACL, DEFACL, !GROUPLIST, Chapter 2, "User Groups."

Why Use It?

Use the GROUPLIST command to add a group to your group list — providing access to files that your username alone will not provide. As mentioned above, you can learn the names of groups on your system from the system manager or by typing FILESTATUS :GROUPS:+. You can also use GROUPLIST to display your group list or add or remove groups from it. User groups are further described in Chapter 2 and in *Managing AOS/VS and AOS/VS II.*

GROUPLIST 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. This section also explains the CLI32 switch /STR=.

/INSERT=n	Adds one or more groups to your group list. The CLI inserts the group name(s) you specify before the group name that has position n in the current group list.
/KILL	Deletes all groups from your group list.
/LEVEL=n	Sets the group list to that 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, 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 your group list to the one used in the previous CLI environment level. With *=n* (CLI32 only), sets your group list to the one used in environment level *n* (0 or greater). The *n* specifies the number of levels above the current level (toward 0).
/PREVIOUS*[=n]*	(CLI32 only.) Same as /P.

GROUPLIST Example

This example assumes that your username is CSTONE, that you are a member of group FUTURES, and that users with username JKM and CSTONE are also members of group FUTURES. (The same example was shown in Chapter 2.)

To begin, you log on.

```
) DIR :UDD:FUTURES ↵          (You make FUTURES the working directory.)
) FILES/AS/S ↵                (Try to list files.)
Error: Read access is required   (Receive access error message.)
) GROUPLIST ↵                 (Check your group list.)
                              (It is empty.)
) GROUPLIST FUTURES ↵         (Join group FUTURES.)
) FILES/AS/S ↵                (Again try to list files.)
PHASE.STAFFING                (You have joined the group, so the system obeys
...                           the FILESTATUS command and lists files.)
) SED PHASE2.OVERVIEW ↵       (Run the SED editor to create and edit a file.)
...                           (Edit the file and exit from SED.)
) ACL/V PHASE2.OVERVIEW↵                  (Check the ACL of the file.)
PHASE2.OVERVIEW CSTONE,OWARE              (Your default ACL does not allow
                                          access to other group members.)
) ACL/V PHASE2.OVERVIEW [!USERNAME],OWARE &↵
&) JKM:FUTURES,WARE +:FUTURES,RE +,E↵     (Add access for group
                                          members: WARE for JKM and
                                          RE for other members.)
) DEFACL PHASE2.OVERVIEW [!USERNAME],OWARE &↵
&) JKM:FUTURES,WARE +:FUTURES,RE +,E↵     (Change your default ACL to allow
                                          group access to other files you create
                                          during this session. You could put
                                          this DEFACL command after the
                                          GROUPLIST command in a macro;
                                          you would execute the macro when
                                          you wanted to join the group.)
...                                       (You continue to work in the group
                                          directory structure.)
) DIR/I MEMOS↵                            (Return to your user directory tree.)
) GROUPLIST/K ↵                           (Set your group list to null – preventing
                                          access to group files to which your
                                          username alone does not grant access.)
) DEFACL [!USERNAME],OWARE +,E ↵          (Restore your original default ACL. You
                                          could put the GROUPLIST/K command
                                          and this one in a macro to executed
                                          when you want to leave the group.)
...                                       (You continue to work in your user
                                          directory tree.)
```

!GROUPLIST *Pseudomacro*

Expands to your current group list (CLI32 with AOS/VS II only).

Format

[!GROUPLIST]

This pseudomacro returns the group list of your current CLI environment.

- No templates.
- No argument switches.
- Macro name switches (described later).
- Requirement: *Standard.*
- See also: GROUPLIST (to display or set your group list), ACL, DEFACL.

Why Use It?

The !GROUPLIST pseudomacro returns your current group list. You might want to use it in a macro to check the group list and take action depending on the list contents.

Macro Name Switches

/LEVEL=n	Expands to the group list established at level nn. 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*, expands to the group list used in the previous CLI environment level. With =*n* (CLI32 only), expands to the group list used in environment level *n* (0 or greater). The *n* specifies the number of levels above the current level (toward 0).
/PREVIOUS*[=n]*	(CLI32 only.) Same as /P.

!GROUPLIST Example

```
) WRITE The current group list is [!GROUPLIST] ↵
The current group list is CONTINGENCIES,STRATEGIC_90S,FUTURES
```

HELP *Command*

Displays information about a CLI command, pseudomacro, or a related topic.

Format

HELP *[item] [...]*

This command displays either a list of topics for which help information is available, or lists information for the specified topic(s).

To start and stop display, use the HOLD function key (or CTRL–S and CTRL–Q sequences). You can stop the help display by pressing CTRL–C CTRL–A.

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

Why Use It?

Use the HELP command to get brief information about command syntax, command usage, or other CLI concepts.

You can design your own help messages as explained in the *Installing* manual for your operating system.

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=.

/V — Displays a detailed description of the item; by default, the CLI displays only a brief description.

/VERIFY — (CLI32 only.) Same as /V.

To display in page mode, turn page mode on as follows before requesting help.

) CHARACTERISTICS/PM ↵

You can later turn page mode off with the command

) CHARACTERISTICS/OFF/PM ↵

HELP Example 1

The following command displays the help topics that are available on the system. (The topics on your system may differ from those shown here.)

) HELP ↵

```
Topics are:

*1_SWITCH       *2_SWITCH       *AFTER_SWITCH     *BROWSE          *CEO_SECRETARY
*CLI_INPUT      *COMMANDS       *CONDITIONALS     *CONTROL_CHARS   *CPIO
*CPIO_VS        *CURSOR_CONTROL *DBMS             *DBNET           *DEBUG
*DISPLAY        *DT             *DUMP_II          ENVIRONMENT      *EXCEPTIONS
*EXEC           *FED            *FILCOM           *FILENAMES       *FILE_TYPES
*FILTER         *GENERIC_FILES  *HELPV            *I_SWITCH        *LABEL
*LDUINFO        *LFE            *LINK             *LINKS           *LOAD_II
*LOGCALLS       *L_SWITCH       *MACROS           *MASM            *MASM_PSEUDO_OPS
*MERGE          *M_SWITCH       *NEWLINE          *NFSCACHE        *NFSMONITOR
*NFSSTART       *NFSSTAT        *NFSTERM          *PATCH           *PATHNAMES
*PED            *POLISHER       *PSEUDO-MACROS    *P_SWITCH        *QUEUES
*Q_SWITCH       *REPORT         *SCOM             *SED             *SLATE
*SORT           *SPRED          *SWITCHES         *TAR             *TAR_VS
*TEMPLATES      *TOPICS         *UPDATE           *VSGEN           *YPCAT
*YPSET          *YPWHICH

For more help about any item above, type 'HELP *item'
```

You can get more help on any item by typing HELP in the format

HELP *item

as in

) HELP *SED ↵

HELP Example 2

This command requests brief help information for the MOUNT command.

) HELP MOUNT ↵

MOUNT – Requires argument(s)
Switches: /1= /2= /L(=) /Q /STR= /AFTER= /DENSITY= /DIRECTORY= /EXTEND /IBM /HOLD /MEDIA= /NOPEND /NOTIFY /QPRIORITY= /READONLY /STRING /VOLID=

HELPV.CLI *Macro*

Displays detailed information about a CLI command, pseudomacro, or topic.

Format

HELPV command

This macro displays the verbose (detailed) help message for the specified command. It works the same way as the HELP/V command. Use the HOLD function key (or CTRL–S and CTRL–Q sequences) to stop and start display. (You can stop the help display by pressing CTRL–C CTRL–A.)

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

Why Use It?

Use the HELPV macro to view detailed information about a CLI command or topic. At one point, HELPV displayed information in page mode. It no longer does so (it displays information the same way as the HELP/V command) but has been retained for compatibility with previous revisions.

HELPV Example

```
) HELPV COPY ↵

                    COPY Command
Format:
COPY destination-pathname source-pathname  [source-pathname ...]

Copies files named in source-pathname to the file named in
   ...
```

!HID *Pseudomacro*

Expands to a host ID.

Format

[!HID *[hostname]*]

This pseudomacro returns the ID number of the local host or of a host specified by its valid host name. The ID number contains no leading zeros.

- No templates.
- No argument switches.
- No macro name switches.
- Requirement: *Standard.*
- See also: HOST, !HOST, SYSID, SYSINFO.

Why Use It?

Use the !HID pseudomacro to include the ID number of a host computer within an argument to a CLI command or macro.

!HID Example 1

```
) WRITE The local host ID is [!HID].
```
The local host ID is 32641.

This command displays a message that reports the ID of the local host.

!HID Example 2

(within a macro)

```
write The host ID for host name %1% is [!hid %1%].
```

This macro accepts a host name argument, and then displays the corresponding host ID. The output will look something like this:

The host ID for host name VEGA is 24335.

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.

HISTORY Example 1

The following dialog shows the HISTORY, HISTORY/SAVE, and PREFIX/HISTORY commands just after user MARILYN has logged on.

```
) WHO ↵
PID:   34 MARILYN      CON4        :CLI.PR

) HISTORY/SAVE=5 ↵

) HISTORY ↵
1 WHO
2 HISTORY/SAVE=5
3 HISTORY

) WRITE Hello ↵
Hello

) SEARCHLIST ↵
:UDD1:MARILYN:MACROS,:UTIL

) DEFACL ↵
MARILYN,OWARE +,RE

) HISTORY ↵
3 HISTORY
4 WRITE Hello
5 SEARCHLIST
6 DEFACL
7 HISTORY

) PREFIX/HISTORY=ON ↵

9!) DIRECTORY ↵
:UDD1:MARILYN:DOCUMENTATION

10!) HISTORY ↵
 6 DEFACL
 7 HISTORY
 8 PREFIX/HISTORY
 9 DIRECTORY
10 HISTORY
11!)
```

HISTORY Example 2

The following dialog shows the HISTORY/KILL, HISTORY/SAVE, HISTORY, HISTORY/RENUMBER, HISTORY/WRITE, and /HISTORY/READ commands.

```
) HISTORY/KILL ↵

) HISTORY/SAVE=5 ↵

) DATE ↵
26-Jul-91

) DEFACL ↵
JENNIFER,OWARE

) TIME ↵
16:56:47

) SPACE ^ ↵
Max   50000, Cur   47674, Rem    2326

) XEQ  MYPROG ↵
Error: File does not exist
XEQ,MYPROG
```

Write the history buffer into file HIST.OUT.

```
) DELETE/2=IGNORE HIST.OUT;  HISTORY/WRITE=HIST.OUT ↵

) WRITE [!UADD 12 15] ↵
27

) DIRECTORY ↵
:UDD1:JENNIFER:CLI

) DATE ↵
26-Jul-91

) TIME ↵
17:00:18

) WRITE [!UMULTIPLY 56 3] ↵
168

) HISTORY ↵
 9 WRITE [!UADD 12 15]
10 DIRECTORY
11 DATE
12 TIME
13 WRITE [!UMULTIPLY 56 3]
14 HISTORY
```

HISTORY Example 2 (continued)

```
) HISTORY/RENUMBER ↵
) HISTORY ↵
2 DATE
3 TIME
4 WRITE [!UMULTIPLY 56 3]
5 HISTORY
6 HISTORY/RENUMBER
7 HISTORY

) TYPE HIST.OUT ↵
DEFACL
TIME
SPACE ^
XEQ MYPROG
```

Write the history buffer into file HIST.OUT, and then read from HIST.OUT

```
) DELETE/2=IGNORE HIST.OUT; HISTORY/WRITE=HIST.OUT ↵
) HISTORY/READ=HIST.OUT ↵
) HISTORY ↵
11 TIME
12 SPACE ^
13 XEQ MYPROG
14 DELETE/2=IGNORE HIST.OUT; HISTORY/WRITE=HIST.OUT
15 HISTORY

) HISTORY/KILL ↵
) HISTORY ↵
1 HISTORY
```

Rename macro file HIST.OUT and execute its commands.

```
) RENAME HIST.OUT HIST.OUT.CLI ↵
) HIST.OUT ↵
JENNIFER,,OWARE
17:02:58
Max   50000, Cur   27678, Rem    22322
Error: File does not exist
XEQ,MYPROG
```

The CLI did not try to execute the fifth and sixth commands in HIST.OUT.CLI because there was an error in the fourth command.

HOST *Command*

Displays a system hostname.

Format

HOST *[host–ID] [...]*

Displays the computer system's hostname, if the system is part of an XTS (XODIAC Transport Services) network. This hostname is the one that was specified to the NETGEN program when the network was generated.

To display your system's hostname, omit arguments.

The host–ID is a unique decimal number assigned to each network node at network generation time. To discover a host–ID, type

) WRITE [!HID] ↵

at a terminal attached to that system. To learn that system's hostname, type HOST, followed by the ID.

- No templates.
- No argument switches.
- Requirement: *Standard.*
- See also: !HID, SYSID, SYSINFO.

Why Use It?

Use the HOST command to find the hostname that corresponds to a host ID.

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=.

/S	Places the hostname in the current environment's CLI String.
/STRING	(CLI32 only.) Same as /S.

HOST Example 1

) HOST ↵
MSIS_01

This command displays the system hostname.

HOST Example 2

```
) HOST 32461 ↵
TZONE
```

This command displays the hostname for the system whose host ID is 32461.

HOST Example 3

These commands display the value of !HID, the local host ID, and then use the result to display the corresponding hostname.

```
) WRITE [!HID] ↵
8037

) HOST 8037 ↵
10000_12N
```

!HOST *Pseudomacro*

Expands to a hostname.

Format

[!HOST *[host-ID]*]

This pseudomacro returns the name of the local host, or the host specified by argument host–ID. The host ID is a decimal number in the range 1 through 32767. The hostname returned is the one that was specified to the NETGEN program when the network was generated.

- No templates.
- No argument switches.
- No macro name switches.
- Requirement: *Standard.*
- See also: !HID, HOST.

Why Use It?

Use the !HOST pseudomacro to include the name of a host computer within an argument to a CLI command or macro.

!HOST Example 1

) WRITE The local host name is [!HOST]. ↵
The local host name is ANTARES.

This command displays a message that reports the name of the local host.

!HOST Example 2

(within a macro)

write The host name for host ID %1% is [!HOST %1%].

This macro accepts a host–ID argument, and then displays the corresponding hostname. The output will look something like this:

The host name for host ID 14 is RIGEL.

!IMPLODE *Pseudomacro*

Removes space and tab separators between arguments (CLI32 only).

Format

[!IMPLODE *[argument] [...]*]

This pseudomacro expands to the text string consisting of the arguments to the pseudomacro. There are no separators in the text string. If the pseudomacro receives no arguments it expands to the null string.

Why Use It?

Use this pseudomacro to concatenate arguments. Suppose you use the pseudomacro !EXPLODE to get individual characters. You can easily invoke !IMPLODE to bring them together again. This task would be more difficult without !IMPLODE. Also, after using !IMPLODE, you can use other commands and pseudomacros to manipulate characters.

- No macro name switches.
- No argument switches.
- Requirement: *Standard.*
- See also: !EXPLODE, !INDEX, SUBSTRING.

!IMPLODE Example 1

Next are several WRITE statements that show the behavior of pseudomacro !IMPLODE.

```
) WRITE [!IMPLODE A B C] ↵
ABC
) WRITE [!IMPLODE A  B□□□C<TAB><TAB>D] ↵  (□ indicates a space.)
ABCD
) WRITE [!IMPLODE A  B□□□C<TAB><TAB>D,E,,,F] ↵
ABCDEF

) WRITE [!IMPLODE A B CDE F G HIJK ... XY Z] ↵
ABCDEFGHIJK...XYZ
) WRITE [!IMPLODE [!EXPLODE A BC DEFG   H I,J K L,,,,MNOP] ] ↵
ABCDEFGHIJKLMNOP
```

!IMPLODE Example 2

Inside a macro, the command

[!IMPLODE %2–5%]

expands to the equivalent of

%2%%3%%4%%5%

So, if macro TEST_IMPLODE.CLI contains the command

WRITE [!IMPLODE %2–5%]

and you give the command

) TEST_IMPLODE A BC DEF GHIJ KLMNO P↵

the displayed result is

BCDEFGHIJKLMNO

!INDEX *Pseudomacro*

Expands to the location of one string in another string (CLI32 only).

Format

[!INDEX search–string character–string]

This pseudomacro expands to a character position. If search string is in character string, the CLI returns the character position of the first character of search string. Otherwise, the CLI returns zero.

The search is case insensitive. So, for example,

[!INDEX ORANGE, REDGREENBLUEORANGEPURPLE]

and

[!INDEX Orange, REDGREENBLUEORANGEPURPLE]

expand to the same value — 13.

You can use parentheses to group arguments. The parentheses will not affect the displayed location. For example, [!INDEX,(C,D),(A,B,C,D,E)] expands to 5.

Why Use It?

Use this pseudomacro to determine whether or not one string occurs within another string and, if so, where the occurrence begins. Also, you can use !INDEX with other commands and pseudomacros like !STRING, !SUBSTRING, and !VARn to manipulate characters.

- No macro name switches.
- No argument switches.
- Requirement: *Standard*.
- See also: !STRING, !SUBSTRING, VARn, !VARn

!INDEX Example 1

The following WRITE statements show the behavior of pseudomacro !INDEX.

```
) WRITE [!INDEX C DEQECAM] ↵
5
) WRITE [!INDEX C DEQECAMCTCC] ↵
5

) WRITE [!INDEX CAT LIONTIGERCATLYNX] ↵
10
) WRITE [!INDEX Cat LIONTIGERCATLYNX] ↵
10
) WRITE [!INDEX DOG LIONTIGERCATLYNX] ↵
0
```

!INDEX Example 2

```
COMMENT This is macro TEST_INDEX.CLI.

STRING [!READ What is your first name?,]
VAR0 [!INDEX, *[!STRING]*, *TOM*ALICE*JEFF*LISA*CLARK*CINDY*JASON*ERIN*]

[!UNE, [!VAR0], 0]
        COMMENT The contents of the string are in "TOM ... ERIN".
        WRITE You are about to see some special information.
        TYPE MEMO_SPECIAL.007
[!ELSE]
        COMMENT The contents of the string are not in "TOM ... ERIN".
        WRITE You do not have permission to see special information.
[!END]
```

This macro validates a name that has arrived from the keyboard (an instance of search string) against a list of names (an instance of character string) that are allowed to see file MEMO_SPECIAL.007. User Jeff sees this file regardless of the case of his name. That is, he may respond to the macro's prompt with Jeff, JEFF, jeff, etc.

INITIALIZE *Command*

Adds a logical disk unit (LDU) to the working directory (AOS/VS version — AOS/VS II version follows).

Format

INITIALIZE unitname *[...]*

Adds the logical disk unit (LDU) that consists of the disk(s) in unit name(s) to the working directory. (An LDU is a directory, created with the Disk Formatter, that includes one or more physical disks.)

If the LDU spans more than one physical disk, you must specify all disk unit names as arguments.

If the LDU is a mirrored LDU, separate mirrored unit names with an exclamation point (!). If you see a *Not synchronized* error message, use INITIALIZE with the /NOMIRROR switch to initialize the image you want; then use the MIRROR command to start synchronizing the other image. See Example 3.

At most sites, commands to initialize and release LDUs (except for diskettes) are included in the system UP and DOWN macros; the commands aren't typed directly.

After executing this command, the system displays the filename of the logical disk unit.

To release an LDU, use its LDU filename, not unit name(s). You can display the LDU name with the command

) FILESTATUS/TYPE=LDU ⤶

in this directory. Or to release all initialized LDUs, you can shut down the system.

- No templates.
- No argument switches.
- Requirement: *Standard,* provided you have Owner access to the LDU (given with the Disk Formatter), Read access to the specified unit name(s) in :PER, and Write access to the working directory); otherwise *Superuser.*
- See also: RELEASE, MIRROR, LDUINFO

Why Use It?

Use the INITIALIZE command to add a logical disk unit to the working directory. You can include this command in your system's UP.CLI macro.

INITIALIALIZE, AOS/VS Version (continued)

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=.

/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	Does not display the filename of the LDU; instead, stores the name in the current environment's CLI String.
/STRING	(CLI32 only.) Same as /S.

INITIALIZE (AOS/VS) Example 1

Assume that someone has used the Disk Formatter to create a single–disk LDU named UDD in disk unit DPJ1. To add the LDU to the root directory, type

```
) SUPERUSER ON; DIRECTORY : ⤶
*) INITIALIZE @DPJ1 ⤶          (For CLI32, the Superuser prompt is Su)
UDD1                           (System displays the LDU filename)
```

INITIALIZE (AOS/VS) Example 2

```
) SUPERUSER ON; DIRECTORY : ⤶
*) INITIALIZE/L=LDU_FILENAME @DPJ1 ⤶
```

In this example, the command stores the LDU name in a file called LDU_NAME. Later, you can release the LDU by using the contents of the disk file, as with

```
*) RELEASE [LDU_FILENAME] ⤶
```

INITIALIZE (AOS/VS) Example 3

Assume the system has single–disk mirrored LDU images, with filename UDD1, in disk units DPJ1 and DPJ2. You could use the following commands to initialize the mirrored LDU.

```
) SUPERUSER ON; DIRECTORY : ⤶
*) INITIALIZE @DPJ1!@DPJ2 ⤶
UDD1
```

The second command adds the mirrored LDU in units DPJ1 and DPJ2 to the root directory. The system displays the LDU name. If the images were not synchronized, the system would reject the command with the error message *Mirrored LDU is not synchronized*. You could then initialize one image, the image in DPJ1, by typing

```
*) INITIALIZE/NOMIRROR @DPJ1 ⤶
UDD1
```

INITIALIZE *Command*

Adds a logical disk unit (LDU) to the working directory (AOS/VS II version — AOS/VS version precedes).

Format

INITIALIZE unitname *[unitname ...]*

or

INITIALIZE unitname!unitname *[unitname!unitname ...]*

Adds the logical disk unit (LDU) that exists on the disk(s) in the unit name(s) to the file system. The LDU becomes a directory beneath the working directory unless you specify another directory with /DIR=. The system will try to use hardware mirroring if possible; to specify software mirroring, use the /NOHARDWARE switch.

If the LDU spans more than one physical disk, you must specify all disk unit names. (If not, you will get an *Incomplete LDU* error message.)

To initialize an LDU and start mirroring it to another LDU, separate the primary and secondary unit names with ! (exclamation point). The LDU images must be synchronized. If the system can mirror the images, it will confirm with a *hardware mirrored* or *software mirrored* status message. If you see a *not synchronized* error message, use INITIALIZE with the /NOMIRROR switch for the LDU you want; then use the MIRROR command to start synchronizing the second image.

If there is more than one LDU on a physical disk, you must specify the LDU name, unique ID, and unit name. Use the form

INITIALIZE/LDUNAME=filename unique_ID/unitname*[/unitname]...*

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

) INITIALIZE/LDUNAME=DATA1 DATA1.IMAGE1/@DPJ11 ⤶

If this LDU spanned two disks, in DPJ11 and DPJ12, you would type

) INITIALIZE/LDUNAME=DATA1 DATA1.IMAGE1/@DPJ11/@DPJ12 ⤶

You can learn which LDUs are on which physical disks using the LDUINFO utility (described later in this chapter) or Disk Jockey, View LDU Information screen (by keyword LDINFO). LDUINFO is also explained in *Managing AOS/VS and AOS/VS II*.

To release an LDU, use its LDU filename, not unit name(s). You can display the LDU filename by typing

) FILESTATUS/TYPE=LDU ⤶

in this directory. Or, to release all initialized LDUs, you can shut down the system.

INITIALIZE, AOS/VS II Version (continued)

- No templates.
- No argument switches.
- Requirement: *Standard,* (provided you have Owner access to the LDU (given via Disk Jockey), Read and Write access to the specified units in :PER, and Write access to the working directory); otherwise *Superuser.*
- See also: RELEASE, LDUINFO (also described in *Managing AOS/VS and AOS/VS II*), MIRROR.

Why Use It?

Use the INITIALIZE command to add a logical disk unit to the working directory.

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=.

/CACHE

Enables the LDU to use data caching if caching has been enabled (at system generation or system startup).

The LDU data cache is an optional area of main memory that can hold disk–based information from LDUs. AOS/VS II may be able to gain access to LDU information much more quickly in the cache than on a disk, so performance could improve.

If, during system generation (VSGEN) or at system startup, memory was allocated for an LDU data cache, you can specify the /CACHE switch for the LDU you are initializing. A sample command is

Su) INITIALIZE/CACHE MY_LDU ↵

You can issue INITIALIZE/CACHE to each LDU whose information you want cached.

The system root LDU is a special case. Since you cannot use the INITIALIZE command on the root LDU, the only time to specify caching for the root LDU is at VSGEN or system startup (by overriding default specs).

/DIR=directory–pathname

Specifies the directory in which you want the LDU initialized. The LDU will be accessible as a directory within that directory. By default, the LDU is initialized in the working directory.

INITIALIZE, AOS/VS II Version (continued)

/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
```

INITIALIZE (AOS/VS II) Example 5

Suppose there is an unsynchronized, mirrored LDU named UDD with images in units DPJ1 and DPJ2. The following sequence shows the messages that return when a person tries to:

- initialize and mirror this LDU (this fails because the images aren't synchronized),
- then initialize one image without /NOMIRROR, and
- finally issue the correct command.

The person then starts synchronizing the second image using the MIRROR command.

```
) SUPERUSER ON ↵
Su) INITIALIZE @DPJ1!@DPJ2 ↵          (Try to mirror.)
Error: Mirror was broken, synchronization is incomplete

Su) INITIALIZE @DPJ1 ↵                (Try one image.)
Error: Incomplete mirrored LDU specified   (Error.)

Su) INITIALIZE/NOMIRROR @DPJ1 ↵       (Use /NOMIRROR.)
UDD2                                  (Success.)

Su) MIRROR/SYNC UDD2 @DPJ2 ↵          (Use MIRROR command to start
                                       synchronization.)
```

INITIALIZE (AOS/VS II) Example 6

There are two LDUs on unit DPJ2, filenames XXX (unique ID XXX.IMAGE1) and YYY (unique ID YYY.IMAGE1). They are mirrored on unit DPJ12, with the unique IDs XXX.IMAGE2 and YYY.IMAGE2. The following commands initialize, and start mirroring on, both LDUs.

```
) SUPERUSER ON ↵
Su) INITIALIZE/LDUNAME=XXX XXX.IMAGE1/@DPJ2!XXX.IMAGE2/@DPJ12 ↵
XXX                                   (System confirms with LDU filename and status.)
From system:
LDU 'XXX', image 'XXX.IMAGE2' is hardware mirrored — synchronized

Su) INITIALIZE/LDUNAME=YYY YYY.IMAGE1/@DPJ2!YYY.IMAGE2/@DPJ12 ↵
YYY                                   (System confirms with LDU filename and status.)
From system:
LDU 'YYY', image 'YYY.IMAGE2' is hardware mirrored — synchronized
```

INITIALIZE (AOS/VS II) Example 7

The following command initializes, and starts software mirroring, the LDU images in units DPJ30 and DPJ31. The LDU filename is UDD4; the LDU unique IDs are UDD4.IMAGE1 (DPJ30) and UDD4.IMAG2 (DPJ31).

```
) SUPERUSER ON ↵
Su) INITIALIZE/NOHARDWARE  @DPJ30!@DPJ31 ↵
UDD4                              (System confirms with LDU filename and status)
From system:
LDU 'UDD4', image 'UDD4.IMAGE2' is software mirrored — synchronized
```

INITIALIZE (AOS/VS II) Example 8

The following command initializes and starts mirroring on three images, in DPJ20, DPJ21, and DPJ22. The LDU filename is DB. The LDU Unique IDs are DB.IMAGE1 (DPJ20), DB.IMAGE2 (DPJ21), and DB.IMAGE3 (DPJ22).

```
) SUPERUSER ON ↵
Su) INITIALIZE  @DPJ20!DPJ21!DPJ22 ↵     (Initialize three images)

DB                                        (System confirms with LDU filename
From system:                              and status)
LDU 'DB', image 'DB.IMAGE2' is hardware mirrored — synchronized
From system:
LDU 'DB', image 'DB.IMAGE3' is hardware mirrored — synchronized
```

JPINITIALIZE *Command*

Initializes a job processor.

Format

JPINITIALIZE n

Some systems have more than one job processor. This command initializes the specified job processor (other than the default job processor, which is usually number 0). The operating system gains control of the job processor once it is initialized.

- No templates.
- No argument switches.
- Requirement: *PID 2* or *System Manager*.
- See also: JPRELEASE.

Why Use It?

Use the JPINITIALIZE command to initialize a job processor as part of bringing up the system. You can include the JPINITIALIZE command in your system UP.CLI macro. If your process in not PID 2, it must have System Manager mode turned on (CLI32 only). This command should precede any PROCESS or XEQ command.

At most sites, commands to initialize and release LDUs (except for diskettes) are included in the system UP and DOWN macros; the commands aren't typed directly.

Loading Microcode

If you omit switches, the system loads the default microcode file into the job processor. If microcode has already been loaded, it is replaced with the default microcode.

To use microcode that has already been loaded into the processor, use the /EXISTING switch to prevent the system from loading new microcode.

To load a specific microcode file (other than the default), include the switch /MCODEFILE= with the pathname of the microcode file you want to load.

JPINITIALIZE 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=.

/EXISTING — Uses the existing microcode (saving time on warm starts). An error occurs if no microcode has been loaded.

/MCODEFILE=pathname — Loads the specified microcode file (instead of using existing microcode or loading the default microcode file).

JPINITIALIZE Example 1

(within the system UP.CLI macro)

```
) JPINITIALIZE 1 ↵
```

This command adds the data processing power of job processor 1 to the computer system.

JPINITIALIZE Example 2

```
) JPINITIALIZE/EXISTING 1 ↵
```

This command initializes job processor 1, using the existing microcode. The /EXISTING switch prevents the reloading of the default microcode file, thereby saving time on a warm start.

JPRELEASE *Command*

Releases an initialized job processor.

Format

JPRELEASE n

This command releases an initialized job processor other than the default job processor. (Job processor is another term for central processing unit, CPU.) The concept of job processors other than the default is relevant only on a system with more than one job processor, like an ECLIPSE MV/20000 Model 2.

(You can initialize a job processor with the JPINITIALIZE command.)

- No templates.
- No argument switches.
- Requirement: *PID 2* or *System Manager.*
- See also: JPINITIALIZE.

Why Use It?

Use this command to remove a job processor from the system without shutting down. (Normal shutdown releases all job processors.) If your process in not PID 2, it must have System Manager mode turned on (CLI32 only).

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=.

/LAST — Informs AOS/VS or AOS/VS II that this is the last job processor connected to a logical processor. (A logical processor is a scheduling construct. It can be defined and connected to one or more job processors by the optional CLASP utility or a user program.)

If a logical processor has been defined and connected to job processor n, and n is the last job processor that is connected to it, you must include this switch to avoid an error message.

JPRELEASE Examples

) JPRELEASE 1 ↵ — Release job processor 1, removing its data processing power from the system.

) JPRELEASE/LAST 2 ↵ — Release job processor 2, which is the last job processor connected to a defined logical processor.

LABEL *Utility*

Writes a volume label to the beginning of a tape or diskette.

Format

XEQ LABEL unitname volid

The LABEL utility writes a label to the beginning of a magnetic tape or diskette, allowing the tape or diskette to be used for labeled access. If a tape is already labeled, relabeling effectively scratches it, preventing access to data already on the tape. (You can access labeled diskettes with the DUMP/LOAD commands under CLI16 only.)

The unitname refers to the tape or diskette unit (@MTB0, @MTC0, or @DPJ10, for example). The volid is the volume ID you assign the tape or diskette. A volid consists of from one through six valid filename characters. Later when using labeled access, you will specify the tape or diskette you want to use by its volid.

The new label contains the volume ID (volid) and empty fields for other information. Later, when someone writes to the media (perhaps for backup, with the DUMP_II command), the system will fill the empty fields with information such as the labeled media filename, file sequence number, creation and expiration date, and operating system name. Thus, the label serves to identify the labeled tape file after the file has been created.

To access the labeled media, use a MOUNT command that specifies the volume ID. Then someone will need to mount the media and specify the unitname on which it's mounted.

The advantages of labeled media over unlabeled media are

- the label information is physically part of the media; the tapes/diskettes themselves contain filename and expiration date.
- labeled tapes can be read on other operating systems; for example, you can create tapes to be read on an IBM system or system that uses ANSI–standard labeled tapes.

By default, labels are created and tapes/diskettes are written in DG format, to be read on an AOS/VS or AOS/VS II–compatible system.

To create tapes to be read on an IBM system (EBCDIC), use LABEL with the /I switch. You can then write to the tape using the DUMP or DUMP_II program with the /IBM switch. You can read tapes created on an IBM (EBCDIC) system using the LOAD/LOAD_II program with the /IBM switch.

To create tapes to be read on an ANSI–based, non–Data General system, use LABEL without the /I switch. To write to the tape, you must use a program that opens and writes to the tape using system calls in ANSI format.

LABEL (continued)

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.

LABEL (continued)

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.)
```

LDUINFO *Utility*

Displays disk unit and LDU status information (AOS/VS II only)

Format

XEQ LDUINFO *[disk-unit-name* or *ldu-filename] [...]*

The LDUINFO utility displays physical and logical information about disk units, including disk unit name, LDU filename, LDU data caching information, LDU unique ID, LDU size, time initialized, parameters like file data element and index element sizes, and mirror image information (see examples).

For information on one or more disk units or LDUs, include the unit or LDU names as arguments. LDUINFO will display the information on your terminal write it or to the file you specify with /L=pathname.

For information on all disk units and LDUs, omit arguments. The utility will prompt for a specific or full report. If you choose a full report (F), the program will request a pathname for an LDU report and full report, generate the report(s), and terminate. If you choose a specific report, the program will display a small menu that lets you select items with function keys. From that menu, the program offers on-line Help.

The LDUINFO program makes extensive use of function keys; you must run it on a video display terminal. Generally, LDUINFO function keys work the same way as other AOS/VS II utility programs such as VSGEN; you can use the AOS/VS and AOS/VS II utilities template with LDUINFO. Use Shift-F1 to get help as needed.

The LDUINFO program file, LDUINFO.PR, is shipped in directory :UTIL. For more details on it, see *Managing AOS/VS and AOS/VS II.*

- Does not accept templates.
- Accepts utility switch (described later).
- Requirement: At least Read and Execute access to the disk entry(ies) in PER or to the LDU(s).
- See also: INITIALIZE, RELEASE, MIRROR.

Why Use It?

Use LDUINFO when you need information on the state of an LDU or disk unit: LDU mirror status, how many disk units an LDU occupies, whether it was initialized for data caching, the LDU unique ID, number of reads, writes, and cache statistics, or other information.

LDUINFO Switch

/L=pathname or /L — Use this when you run LDUINFO without arguments to write information to file pathname. Omit the pathname to use the generic @LIST file. If the pathname does not exist, LDUINFO creates it; if it does exist, LDUINFO appends to it.

LDUINFO Example

The following example shows LDUINFO reporting on disk unit DPJ0.

```
Su) LDUINFO @DPJ0 :↵                          (Specifies disk unit and LDU
                                               filename.)

@dpj0                                         (Disk unit report begins.)

        Initialized full path:  :
        LDU:                    ROOT
        LDU Unique ID:          ROOT.IMAGE1

Piece 1 of 1                  System area ID: 500
Size: 1159130 blocks          Channel number: 0
BBT size: 256 blocks          Device code: 64
Starting LDU address: 0       Unit number: 0
Unit type: DPJ                Modify time: ...    (Disk unit report ends.)

ROOT [cached]                                 (LDU report begins.)
Size: 1159130 blocks
Cache statistics:
        Reads: 172521         Read hits:    99190
        Writes: 89235         Write hits:   72202

Time initialized: Mon Mar 25 ...

Default LDU parameters:
        Primary element size:          4    Secondary element size:   4
        Number of pirmary elements:    1    Index element size:       1
        Maximum index levels:          3

Active image(s):
   ROOT.IMAGE1 - DPJ0                         (LDU report ends.)
```

!LENGTH *Pseudomacro*

Expands to the lengths (in characters) of the arguments (CLI32 only).

Format

[!LENGTH *[argument [...]]*]

This pseudomacro expands to the character lengths of each of the arguments. If there are no arguments, the pseudomacro expands to zero.

Why Use It?

Use this pseudomacro to determine the number of characters in each word of a text string. Also, !LENGTH works with other commands and pseudomacros to manipulate characters.

- No macro name switches.
- No argument switches.
- Requirement: *Standard.*
- See also: !IMPLODE, !INDEX, STRING, !SUBSTRING, VARn.

!LENGTH Examples

Next are several WRITE statements that show the behavior of pseudomacro !LENGTH.

```
) WRITE [!LENGTH ABCDEF] ↵
6

) WRITE [!LENGTH ] ↵
0

) WRITE [!LENGTH OF A TEXT STRING] ↵
2 1 4 6

) WRITE [!LENGTH Here are,, some,,,commas for null strings.] ↵
4 3 0 4 0 0 6 3 4 8

) WRITE [!DATE] [!LENGTH [!DATE]] ↵
21-JUL-89 9
```

LEVEL *Command*

Displays the number of the current CLI environment level.

Format

LEVEL

This command displays your current environment level number.

- No arguments.
- Requirement: *Standard.*
- See also: POP, PUSH, !LEVEL.

Why Use It?

Use the LEVEL command to display the number of the current environment level. You frequently use this command along with the PUSH and POP commands.

Each CLI environment level can contain different settings. If what you do depends on the current settings, you can use the LEVEL command to determine the setting before taking action.

A macro or program that terminates unexpectedly may not return you to the environment level in which you started the macro or program. When this happens, you can use the LEVEL command to ascertain whether or not you have returned to the starting environment level.

NOTE: Within a macro, you can obtain the current level number by using the !LEVEL pseudomacro.

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=.

LEVEL Example 1

```
) LEVEL ↵
Level 3

) POP/V ↵
Level 2
```

The first command displays the current environment level. The second command moves down to the previous level, and verifies the move.

LEVEL Example 2

```
) LEVEL ↵                    (Displays the current level.)
Level 0                      (It is 0.)

) DIR ↵
:UDD:JILL:QQQ                (Displays the working directory.)

) PUSH ↵                     (Pushes down a level.)

) LEVEL ↵                    (Displays the current level.)
Level 1                      (It is 1.)

) DIR :UDD:JILL:XXX ↵
...                          (Goes from :UDD:JILL:QQQ to another directory.)
...                          (Executes commands in that directory.)
...

) LEVEL ↵                    (Displays the current level.)
Level 1                      (It is 1.)

) POP ↵                      (Returns to the previous level's environment.)

) DIR ↵
:UDD:JILL:QQQ                (Displays the working directory.)

) LEVEL ↵                    (Displays the current level.)
0
```

!LEVEL *Pseudomacro*

Expands to the current CLI environment level number.

Format

[!LEVEL]

This pseudomacro represents the level number of the current CLI environment. The base level is level 0.

- No arguments.
- No macro name switches.
- Requirement: *Standard.*
- See also: LEVEL, POP, PUSH.

Why Use It?

Use the !LEVEL pseudomacro to include the current level number within an argument to a CLI command or pseudomacro. You might want to check the current level before taking a particular action. In the example, the macro uses the !LEVEL pseudomacro to determine when the current level is 0.

!LEVEL Example 1

The macro POPBACK.CLI contains the following lines:

```
comment This is macro POPBACK.CLI.
[!ugt, [!level], 0]
        pop
        %0%
[!end]
```

This macro checks the current CLI environment level. If the level number is greater than 0, the macro moves down a level and calls the macro again. When the level number is equal to zero, the macro does nothing.

!LEVEL Example 2

```
) WRITE THE CURRENT LEVEL IS [!LEVEL] ↵
THE CURRENT LEVEL IS 0

) PUSH; PUSH; WRITE NOW THE CURRENT LEVEL IS [!LEVEL] ↵
NOW THE CURRENT LEVEL IS 2
```

LISTFILE *Command*

Displays or sets the current list file pathname.

Format

LISTFILE *[pathname]*

Displays (without the pathname argument) the current list file setting, or sets it to the pathname specified. If you have not set the list file, the CLI returns @LIST, the generic list file, since there is no list file setting, except in batch mode. When you set the list file to pathname, the CLI creates the file (if it does not exist) or appends subsequent data to it (if it does exist).

The CLI passes the list file to a process created by an EXECUTE or XEQ command. This feature is useful when you want to write data to different files. When coding a program that must open and write to a file, you can use @LIST within your program instead of the file pathname. Then, before you run the program, set LISTFILE to the designated pathname. The CLI passes pathname to the new process as its generic list file.

NOTE: By default, the system writes to the list file whenever the pertinent buffer is full or when the list file is closed (as when the program that uses it terminates). To force the system to write to the list file whenever the program writes to the list file, use the /FORCE switch (CLI32 only). To force output to the list file with CLI16, close the list file (LISTFILE/K) or, if you are running a program, wait until the program terminates.

- No templates.
- No argument switches.
- Requirement: *Standard.*
- See also: !LISTFILE, DATAFILE.

Why Use It?

Use the LISTFILE command to designate a file to be the current environment's list file. You may want to assign a file to be used generically for @LIST. Assigning a file to @LIST lets you gather output from CLI commands by using the /L switch without having to specify a filename.

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=.

/G	Sets the filename to the generic filename @LIST. (Omit the pathname argument.) Do not use together with /K.
/GENERIC	(CLI32 only). Same as /G.

LISTFILE (continued)

/FORCE	(CLI32 only.) Tells the system to force output to the list file every time the program writes to it. Normally, the system retains output in memory, without writing it to disk, until one of the following occurs: 1) the program has written 512 characters to the list file; or 2) the list file is closed (as when the program terminates). Use this switch if you need to have the list file remain synchronized with program execution.
/K	Sets the list file to a null string. (Omit pathname argument.) Do not use together with /G.
/KILL	(CLI32 only). Same as /K.
/LEVEL=n	(CLI32 only.) Sets the list file to that 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 list file to the pathname used in the previous environment. (Omit the pathname argument.) With *=n* (CLI32 only), sets the list file to the pathname 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.

LISTFILE Example 1

```
) LISTFILE ↵
@LIST
) LISTFILE  LIST.OUT ↵
```

This command displays the current list file setting, and then sets the list file to LIST.OUT. The system creates LIST.OUT if it does not already exist.

LISTFILE Example 2

```
) DATAFILE ↵
@DATA                                  (Displays DATAFILE setting — none.)
) LISTFILE ↵
@LIST                                  (Displays LISTFILE setting — none.)

) DATAFILE  MAY_SALES.IN ↵             (Sets DATAFILE to MAY_SALES.IN.)
) LISTFILE/FORCE MAY_SALES.OUT ↵       (Sets LISTFILE to MAY_SALES.OUT.)
) XEQ  REPORT_PROG ↵                   (Runs REPORT_PROG, taking data
                                        from datafile MAY_SALES, and sending
                                        the output to list file MAY_SALES.OUT.)
```

!LISTFILE *Pseudomacro*

Expands to the full pathname of the current list file.

Format

[!LISTFILE]

This pseudomacro represents the full pathname of the current list file.

- No arguments.
- Accepts a macro name switch (described later).
- Requirement: *Standard.*
- See also: LISTFILE.

Why Use It?

Use the !LISTFILE pseudomacro to include the full pathname of the current list file within a CLI command argument.

Macro Name Switch

/LEVEL=n	(CLI32 only.) Expands to the list file to 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 using /LEVEL over using /PREVIOUS.
/P*[=n]*	Without =*n*, expands to the list file used in the previous environment. (Omit the pathname argument.) With =*n* (CLI32 only), expands to the list file 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.

!LISTFILE Example 1

(within a macro)

```
copy/a [!listfile] %1%
```

This macro statement appends the contents of the file specified in the macro argument to the current list file.

!LISTFILE Example 2

```
) XEQ MYPROGRAM [!LISTFILE] ↵
```

This command executes MYPROGRAM and provides the current list file as an argument to the program.

!LISTFILE Example 3

```
) WRITE The current list file is [!LISTFILE] ↵
The current list file is @LIST

) PUSH ↵
) LISTFILE :UDD:ALICE:WORK999 ↵
) WRITE Now the current list file is [!LISTFILE] ↵
Now the current list file is :UDD:ALICE:WORK999

) WRITE [!LISTFILE/P] ↵
@LIST
```

These commands display the current list file, change the environment by moving down one level, set the new list file for the new environment, display it, and display the list file for the previous environment.

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.

LOAD (continued)

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.

LOAD (continued)

/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.

LOAD (continued)

/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 (continued)

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.

LOAD_II (continued)

/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.

LOAD_II (continued)

/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.

LOCALITY *Command*

Assigns a new user locality to a process.

Format

LOCALITY { process–ID | processname | username:processname } new–user–locality

This command changes a process's user locality to the new user locality you specify. (User localities range from 0 through 15.) By changing user locality, you can change the class the process belongs to, which may affect the amount of computer time the process receives. This command is meaningful only if your system has nondefault user localities defined (via the PREDITOR utility) and has classes defined and enabled (via the optional CLASP utility).

Before you can change the locality of your user process or any of its sons, your user profile must allow — via the Use Other Localities privilege — the new user locality. There is a Use Other Localities privilege for both nonbatch and batch operation. The nonbatch privilege allows locality changes via a LOCALITY command typed on your terminal, while the batch privilege allows locality changes via a LOCALITY or PROCESS/LOCALITY command issued in a batch job.

You must supply a process ID or a process name. If you supply a simple process name, the CLI assumes your username. You can use either process name form, but only if the process was created with the PROCESS command and /NAME=name switch.

To discover the user locality of a process, run the PED utility with the /ULOCALITY switch. (To display the program locality of a process, also use PED's /PLOCALITY switch.)

- No templates.
- No argument switches.
- Requirement: *PID 2* or *System Manager* turned on (to change any process's user locality); otherwise, to change the locality of your process or its sons. you need the *Use Other Localities* privilege and privilege to use the new locality in batch or nonbatch.

Why Use It?

By changing user locality, you can change the class the process belongs to, which may affect the amount of computer processing time the process receives.

LOCALITY 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=.

LOCALITY Example

Check your PID.

```
) WHO ↵
PID :  43  JOAN   043  :CLI.PR
```

Run the PED utility to check user locality. Exit via the B command.

```
) PED/PID/USER/PLOCALITY/ULOCALITY/CLASSNAME ↵
```

PID	*USERNAME*	*UL*	*PL*	*CLASSNAME*
1	*PMGR*	*0*	*0*	*DEFAULT.CL*
8	*OP*	*0*	*0*	*DEFAULT.CL*
43	*JOAN*	*0*	*0*	*DEFAULT.CL*

Change user locality for PID 43.

```
) LOCALITY 43 1 ↵
```

Run PED again, which displays a new user locality and class name. Exit via the B command.

```
) PED/PID/USER/PLOCALITY/ULOCALITY/CLASSNAME ↵
```

PID	*USERNAME*	*UL*	*PL*	*CLASSNAME*
1	*PMGR*	*0*	*0*	*DEFAULT.CL*
8	*OP*	*0*	*0*	*DEFAULT.CL*
43	*JOAN*	*1*	*0*	*PRIVILEGED*

LOCK *Command*

Locks the CLI (CLI16 version — CLI32 version follows).

Format

LOCK

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 following commands:

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

Thus, you cannot execute programs from a locked LOCK_CLI. A locked LOCK_CLI also restricts the power of other commands:

ACL	can display, but not set, the access control list for a file.
QPRINT	can print files in the root directory only; the /DELETE switch has no effect.
REVISION	can display, but not set, a program's revision number.
TYPE	The /L and /L= switches have no effect; output goes to the console only.

Also, when it is locked, the CLI automatically turns off the Superuser and Superprocess privilege if either was turned on. The process termination sequences CTRL–C CTRL–B, CTRL–C CTRL–E, and CTRL–D CTRL–D are ignored.

- No arguments.
- No templates.
- Requirement: *Standard* (but usually executing LOCK_CLI requires Superuser; also, see note following).
- See also: UNLOCK.

LOCK, CLI16 Version (continued)

Why Use It?

Use the LOCK command at the system console to prevent the unauthorized use of system resources, and to protect the system and its data from accidental or malicious destruction.

You can include this command in the system UP.CLI macro to lock the CLI automatically.

NOTE: To unlock a locked LOCK_CLI, you must specify a password. If you don't know the password, you cannot exit from this CLI. Setting the LOCK_CLI password is explained in *Managing AOS/VS and AOS/VS II.*

Command Switches

The section "Common CLI Switches," earlier in this chapter, describes the /1, /2, and /Q switches, which you can use with this command. The /L and /L= switches are ineffective.

LOCK (CLI16) Example

```
) LOCK ↵
```

This command locks this CLI process, and allows only a very limited range of operations.

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.

LOCK, CLI32 Version (continued)

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.

LOGFILE *Command*

Displays or sets the user log file.

Format

LOGFILE *[pathname]*

This command either displays the name of your log file, or lets you name a file as your log file.

When you name a file as your log file, the CLI automatically starts logging CLI dialog — commands and responses — to it. The CLI will create the file if it does not exist; if it does exist, dialog will append to it. Logging continues until you turn it off (LOGFILE/K) or this CLI process terminates.

The CLI cannot record user interaction with other programs (for example, with the DISPLAY or Link program).

With CLI32, the log file is part of the environment; you can specify a different prefix for a different level. With CLI16, the log file remains constant through all environment levels.

- No templates.
- No argument switches.
- Requirement: *Standard.*
- See also: HISTORY (CLI32 only), SYSLOG.

Why Use It?

Use the LOGFILE command to keep a record of your interaction with the CLI. After you designate a log file, the system records in that file each command you enter and the system's response to it.

You can also use this command to discover what, if any, log file is being used. To turn off logging, use the /K 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 switch /STR=.

/K	Stops logging dialog and sets the log–file name to null. (Omit the pathname argument.)
/KILL	(CLI32 only.) Same as /K.

LOGFILE (continued)

/LEVEL=n — (CLI32 only.) Sets the log file 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*, uses the log file specified in the previous CLI environment. (Omit the pathname argument.) With =*n* (CLI32 only), uses the log file specified at the specified environment level. The *n* specifies the number of levels above the current level (toward 0).

/PREVIOUS*[=n]* — (CLI32 only.) Same as /P.

/V — Displays the name of the log file. This switch is useful when you designate a log file.

/VERIFY — (CLI32 only.) Same as /V.

LOGFILE Example 1

```
) LOGFILE SEPT.12.90.LOG ↵
```

This command begins logging your CLI session in a file called SEPT.12.90.LOG. If the file does not exist, the CLI creates it; otherwise, the CLI appends information to the existing file.

LOGFILE Example 2

```
) LOGFILE ↵
SEPT.12.90.LOG

) LOGFILE/K ↵
) TYPE  SEPT.12.90.LOG ↵
...
```

This command sequence displays the name of the log file, stops logging (LOGFILE/K), and then types the file.

LOGOFFMACRO *Command*

Sets or displays the log–off macro filename (CLI32 only).

Format

LOGOFFMACRO *[logoff–macro–name]*

With no argument, this command displays the pathname of the current log–off macro. With an argument, this command sets the current log–off macro name to file *logoff–macro–name.*

The LOGOFFMACRO command does not execute the log–off macro. The CLI executes the log–off macro only when you give the BYE command or when you type the macro name. The CLI does not execute this macro if your CLI process terminates abnormally.

If somehow the log–off macro is deleted or renamed after you set it, the CLI will not execute your next BYE command. After you enter BYE, the CLI will display

Cannot access logoffmacro, it has been removed
Error: File does not exist, xxx

Then the CLI will set the logoff macro name to null, so the BYE command will succeed.

- No templates.
- Requirement: *Standard.*
- See also: BYE.

Why Use It?

Use the LOGOFFMACRO command to specify a macro that the CLI will execute when you use the BYE command to end a session with AOS/VS or AOS/VS II. Usually this macro contains instructions to delete temporary files you may have created during your session.

You can set the logoff macro in your logon macro, along with your searchlist and default ACL.

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=.

/KILL — Sets the log–off macro filename to null so that the BYE command will not invoke any macro.

LOGOFFMACRO Example

```
) TYPE  CLEANUP.CLI ↵
DELETE/V  ?+.TMP
WRITE Cleanup complete.

) LOGOFFMACRO CLEANUP.CLI ↵

) LOGOFFMACRO ↵
:UDD:ALICE:MACROS:CLEANUP.CLI

) BYE ↵
Deleted ?TEMPORARY_FILE.TMP
Deleted ?SORT1.TMP
Deleted ?SORT2.TMP
Cleanup complete.
AOS/VS II CLI  Terminating ...
```

!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

MIRROR pathname-of-initialized-image { unitname *[unitname ...]* / unique_ID/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 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.

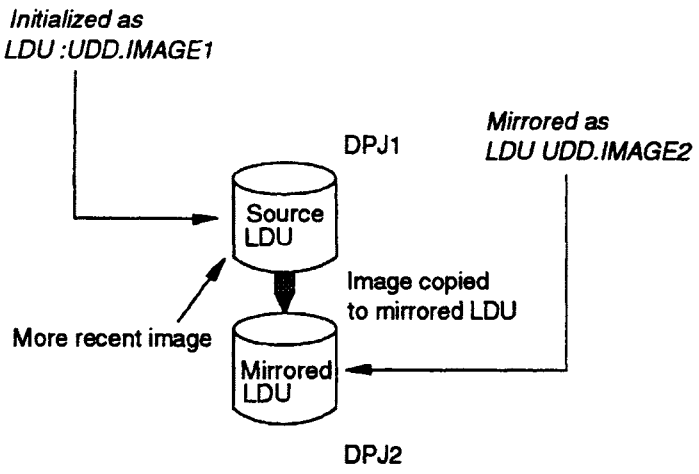
The unitname must contain an LDU image that has the same filename and same size (created with the Disk Jockey program) as the initialized image. As the pathname-of-the-initialized-image, use a full pathname; for example, if an LDU named UDD1 as initialized in the root, its pathname is :UDD1.

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.

MIRROR, AOS/VS II Version (continued)

For example, initialize a newer image on DPJ1 and mirror it on DPJ2:

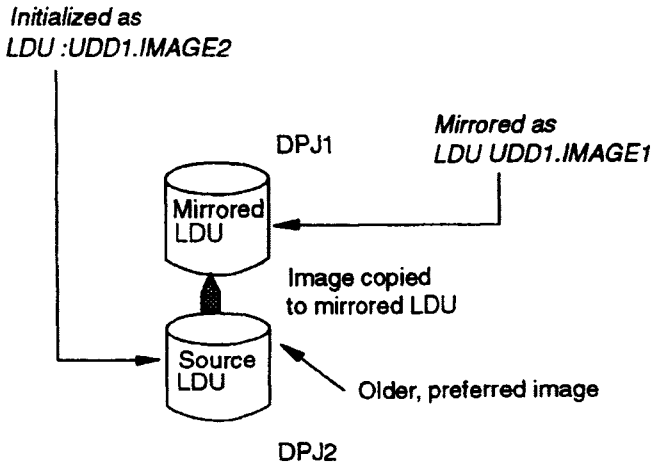
```
Su) INITIALIZE/NOMIRROR @DPJ1 ↵        (Initializes the newer
UDD                                     source image)
Su) MIRROR/SYNC :UDD UDD.IMAGE2/@DPJ2 ↵ (System echoes LDU filename)
                                        (Starts synchronizing the
                                        image 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:

```
Su) INITIALIZE/NOMIRROR @DPJ2 ↵        (Initializes the older
UDD                                     source image)
Su) MIRROR/FORCESYNC :UDD UDD.IMAGE1/@DPJ1 ↵ (System echoes LDU filename)
                                        (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.

MOUNT (continued)

/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.

MOVE *Command*

Moves a copy of one or more files to another directory.

Format

MOVE destination–directory *[sourcefile[[...]*

This command creates a duplicate of the specified file(s) and moves it to the specified destination directory. Normally, all source files must be in or subordinate to the working directory.

You may use templates for the sourcefile arguments. If you omit a sourcefile argument, the CLI assumes the template # and tries to copy all files in and below the working directory — retaining the current directory structure. To move directories and files without retaining the current directory structure, use the /FLAT switch.

The original files are not affected.

- Accepts templates for sourcefile.
- No argument switches.
- Requirement: *Standard.*
- See also: COPY and, for AOS/VS II, RENAME.

Why Use It?

Use the MOVE command to create a copy a file in a different directory under the same name; to replace a file in another directory; or to select files that meet certain criteria.

In AOS/VS II, if you want to move the original file to a different directory (instead of creating a copy there), use the RENAME command to change the pathname of the original 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 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–mmm–yy:hh:mm:ss), date (dd–mmm–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.

MOVE (continued)

/BACKUP — Tells the XODIAC File Transfer Agent (FTA) to use checkpointing when moving copies of files over a communications line. (Meaningful only with the /FTA switch.) If the transfer is aborted, the system returns a Unique Identity (UID) for use with the /BACKUP=uid switch.If you use a template in the sourcefile argument(s), FTA recovers only the file being transferred.

/BACKUP=uid — Tells FTA to use the supplied UID to recover from an aborted file transfer for which the /BACKUP switch was specified. (Meaningful only with the /FTA 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–mmm–yy:hh:mm:ss), date (dd–mmm–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.

/BUFFERSIZE=n — Reads the source file(s) into a buffer of n bytes.

/COMPRESS — Compresses data files (saves transmission time). FTA decompresses files after transfer.

/COUNT — (CLI32 only.) Counts the number of files moved.

/DELETE — Deletes the nondirectory file in the destination directory if it has the same name as the source file. If you include the verify (/V) switch, displays names of deleted files. FTA does not acknowledge files deleted.

/FLAT — Does not maintain the directory hierarchy; moves all source files into the destination directory.

/FTA — Copies the files via FTA, XODIAC's File Transfer Agent. FTA ignores the /BUFFERSIZE and /NACL switches, and does not acknowledge files deleted as a result of the /DELETE or /RECENT switches. With /FTA, only the /DELETE, /RECENT, /V, /COMPRESS, and /BACKUP switches are meaningful. To move a file over a network, you must have the same username/password combination on both hosts (as well as write or append access to the target directory).

/NACL — Gives the default ACL to moved file(s). For network operations, the XODIAC Resource Management Agent (RMA) ignores the /NACL switch when moving files across the network to a remote directory.

/RECENT — If there is a nondirectory file in the destination directory with the same filename as a source file, this switch moves that source file only if it is more recent than the existing file.

MOVE (continued)

/SORT — (CLI32 only.) Sorts alphabetically the filenames this command processes. To see the names, you must also use /V.

/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

```
MOVE/TYPE=\CPD :UDD1:COMMON #:PROJ.–
```

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

```
MOVE/TRAVERSE=\CPD :UDD1:COMMON #:PROJ.–
```

give expected results.

/TYPE=typecode — Selects files of the specified type, where the type code is either a 3–letter mnemonic or a decimal number (0–255) 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 — Lists the name of each file the system moves on @OUTPUT. When used with /DELETE or /RECENT, this switch also verifies each deletion.

MOVE Example 1

```
) MOVE/V  :UDD:COMMON  REPORT+ ↵
REPORT.MAR
REPORTING_PROCEDURES
REPORT.JAN
REPORT.DEC
REPORT.FEB
REPORT_SUMMARY
```

This command copies to the directory :UDD:COMMON all files in the working directory that begin with the characters REPORT. The /V switch displays the files that are moved.

MOVE Example 2

```
) MOVE/V  :UDD:COMMON  REPORT+ ↵
Warning: Filename already exists, File REPORT_SUMMARY

) MOVE/V/RECENT  :UDD:COMMON  REPORT+ ↵
Deleted REPORT_SUMMARY
REPORT_SUMMARY
```

This command sequence resembles the one in Example 1. It tries to move REPORT+ files to :UDD:COMMON, but a file with the name REPORT_SUMMARY already exists there. Adding the /RECENT switch tells the system to move only the most recent version. If a file of the same pathname exists in the destination directory, the system compares times last modified; the file is deleted and replaced only if it is older than the one within the working directory. The /V switch verifies files copied.

MOVE Example 3

```
) MOVE/V/FLAT  DIR2 ↵
.
(displays names)
.
```

This command moves all directories and files in the working directory to directory DIR2. The /FLAT switch tells the system to move directories as if they were files; all files and directories will be moved, without directory structure, to DIR2.

MOVE Example 4

```
) MOVE/V/AFTER/TLM=3-APR-90:12:30  NEW_SOURCES  +.F77  +.C ↵
.
(displays names)
.
```

This command copies all FORTRAN 77 and C programs modified after April 3, 1990 at 12:30 p.m. to directory NEW_SOURCES. It verifies filenames moved.

MOVE Example 5

```
) MOVE/V/RECENT/FTA  :NET:PURCHASING:UDD:CHRIS  REPORTS:# ↵
.
(displays names)
.
```

This command uses the /FTA switch to directory REPORTS and all its files across the network to the directory :UDD:CHRIS on a remote host named PURCHASING. If there is a pathname conflict, the system copies the source file only if it is newer (was modified later) than the existing file.

!NEQUAL *Pseudomacro*

Tests two values for inequality, and continues execution based on the result.

Format

```
[!NEQUAL, argument1, argument2]
        ...
        ...
[ [!ELSE] ]
        ...
        ...
[!END]
```

This pseudomacro compares two arguments, character by character, to determine whether or not they are unequal.

If the arguments are unequal, the CLI executes the statements that follow the !NEQUAL statement up to the corresponding !ELSE or !END.

Otherwise, the CLI ignores the statements up to the corresponding !ELSE or !END statement. If the sequence includes an !ELSE statement, the CLI executes the statements that follow !ELSE up to the corresponding !END.

Note that the !NEQUAL pseudomacro compares the arguments as strings. This means that when [!NEQUAL] compares 01 and 1, it will produce a true condition (because it considers them unequal). To compare the arguments numerically, use the !UNE pseudomacro.

- No arguments.
- No macro name switches.
- Requirement: *Standard.*
- See also: !UNE, !EQUAL, !ELSE, and !END.

Why Use It?

Use the !NEQUAL pseudomacro within a macro to perform one or more actions only if two values are unequal. !NEQUAL is best used for strings; if you want to compare integer values, use !UNE instead.

!NEQUAL Example 1

(within a macro)

```
[!nequal, [!read Do you want to skip printing?,,],Y]
        qprint myfile
[!end]
```

The first statement requests a response and then compares it with the letter Y. If the two are not equal, the macro performs the QPRINT command; otherwise, the macro resumes processing following the !END statement.

!NEQUAL Example 2

(within a macro)

```
[!une, [!read Do you want to skip printing?,,],Y]
        qprint myfile
[!else]
        write File not printed.
[!end]
```

This version of the macro in Example 1 includes an !ELSE statement, which causes the macro to write a message to the screen if the response to the prompt is not the letter Y.

!NEQUAL Example 3

In the next macro statement, the !NEQUAL pseudomacro compares string arguments that include separator characters. You must enclose both arguments in parentheses.

```
[!nequal, ([!string]),(END OF FILE)]
```

!NEQUAL Example 4

In the following two command lines, assume that the variable %1% equals HI. Therefore, in the first command line the !NEQUAL pseudomacro is true. In this case the CLI executes the command between [!NEQUAL] and [!ELSE], and ignores the command between [!ELSE] and [!END].

```
) [!NEQUAL %1% BYE]WRITE NOT EQUAL [!ELSE]WRITE EQUAL [!END] ↵
NOT EQUAL
```

In the second command line the !NEQUAL pseudomacro is false. In this case the CLI executes the command between [!ELSE] and [!END], and ignores the command between [!NEQUAL] and [!ELSE].

```
) [!NEQUAL %1% HI]WRITE NOT EQUAL [!ELSE]WRITE EQUAL [!END] ↵
EQUAL
```

In both command lines there is no space immediately preceding the WRITE command.

!OCTAL *Pseudomacro*

Expands to the octal equivalent of a decimal number.

Format

[!OCTAL decimal-number *[...]*[1]]

[1] Multiple arguments available in CLI32 only.

This pseudomacro returns the octal equivalent of a decimal integer. Argument(s) must be in the range 0 through 4,294,967,295 (resulting in an octal value in the range 0 through 37,777,777,777).

- No templates.
- No argument switches.
- No macro name switches.
- Requirement: *Standard.*
- See also: !DECIMAL.

Why Use It?

Use the !OCTAL pseudomacro to convert a decimal integer to an octal value, and use that value in a CLI command argument.

!OCTAL Example 1

```
) WRITE [!OCTAL 1000] ↵
1750
```

This command displays the octal equivalent of the decimal value 1000.

!OCTAL Example 2

The macro OCTAL.CLI contains the following line:

```
write Decimal %1% is [!octal %1%] octal.
```

The macro takes a decimal integer as an argument, and then displays the octal equivalent. For example,

```
) OCTAL 97 ↵
Decimal 97 is 141 octal.
```

OPEN *Command*

Opens a file for input or output (CLI32 only).

Format

OPEN *[/FILEID=file-ID] [pathname]*

The OPEN command can open a file for reading or writing (not both).

To open a file, include its pathname and, optionally, a file ID with the /FILEID= switch. The file ID is simply a label you will use later to refer to the file in READ, WRITE, and/or CLOSE commands. If you omit the file ID, the CLI assigns a default ID, which is the filename (not pathname), without any trailing suffix. For example, if the pathname is MYDIR:MYMACRO.CLI, the default file ID is MYMACRO.

If you specify a pathname that exists, the file is opened for reading by default. If you will later write text to the file, use the /WRITE switch. If you will later get information from the file with the READ command, use /READ.

If you omit arguments, OPEN displays the pathnames of each file you previously opened with this command along with its

- File ID.
- Reading or writing status.
- String the CLI will return when it encounters the end of the file during a READ command.

When you are finished with the file, close it with the CLOSE command and the file's identifier.

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

Why Use It?

Use the OPEN command before reading from or writing to a file. Or use it to display information about files you have previously opened.

When reading from a file, you can read lines into named strings (/STR= switch), and later write them to another file (with the same named string and /STR= switch).

OPEN 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=.

/APPEND	If the file already exists, appends information to its end. Future WRITE command to the file place information at the end of the file. /APPEND is valid only with the /WRITE switch.
/DELETE	If the file already exists, deletes and recreates the file. /DELETE is only valid with the /WRITE switch.
/EOF=string	Specifies the string the system will return when it encounters the end of the file during a READ command. /EOF is valid only with the /READ switch.
/FILEID=file–ID	Indicates the file–ID. This is a label you will use later in WRITE, READ, and/or CLOSE commands to this file. It must be a valid AOS/VS II filename. If you omit the file ID, the CLI assigns a default ID, which is the filename (not pathname), without any trailing suffix.
/EXCLUSIVE	Opens the file for your exclusive access.
/FORCE	Forces the system to place data in filename every time you issue a 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 only valid with the /WRITE switch.
/READ	Opens the file for reading with future READ statements.
/WRITE	Opens the file for writing with future WRITE/FILE statements.

OPEN Example 1

```
) OPEN/WRITE   MYFILE ↵          (Open MYFILE for writing)
MYFILE                           (CLI returns the file ID)

) WRITE/FILEID=MYFILE  Hello ↵   (Write text to the file)
) CLOSE/FILEID=MYFILE ↵          (Close the file)
) TYPE MYFILE ↵                  (Type the file)
Hello
```

OPEN Example 2

```
) STRING/NAME=INFILE/FILEID=INFILE ↵                    (Put /FILEID= and input
                                                         file ID in a string.)

) OPEN[!STRING/NAME=INFILE]/READ/EOF="" MY_LIST ↵       (Open the input file.)
INFILE

) STRING/NAME=OUTFILE/FILEID=OUTFILE↵                   (Put /FILEID= and output
                                                         file ID in another string.)

) OPEN[!STRING/NAME=OUTFILE]/WRITE MY_OUTPUT ↵          (Open output file.)
OUTFILE

) READ[!STRING/NAME=INFILE]/STR=LINE1 ↵                 (Read from the input file.)

) WRITE[!STRING/NAME=OUTFILE]/STR=LINE1 ↵               (Write to the output file.)

) OPEN ↵
:UDD:LISA:WORK:MY_LIST      INFILE        Read **
:UDD:LISA:WORK:MY_OUTPUT    OUTFILE       Write <none>

) CLOSE/ALL ↵                                           (Close all files)
```

This example shows use of named strings to open input and output files, read a line from the input file, and write it to the output file. The OPEN command without switches or arguments displays open status; and finally, the CLOSE/ALL command closes all open files.

OPEN Example 3

The following statements place three records into a freshly created file, close it, reopen it, and then display the file's records.

```
) OPEN/DELETE/WRITE THREE_RECORDS ↵
THREE_RECORDS
```

This command displays the file's identifier, which is the same as the filename. Future CLOSE, READ, and WRITE/FILE statements will use THREE_RECORDS to identify the file. The command deletes file THREE_RECORDS if it already exists. Then it creates and opens an empty file named THREE_RECORDS.

```
) WRITE/FILEID=THREE_RECORDS  Record 1 of 3. ↵
) WRITE/FILEID=THREE_RECORDS  Record 2 of 3. ↵
) WRITE/FILEID=THREE_RECORDS  Record 3 — and last — of 3. ↵
```

These commands place three records into file THREE_RECORDS.

```
) OPEN ↵
:UDD:ALICE:THREE_RECORDS  THREE_RECORDS    Write <none>

) CLOSE/FILEID=THREE_RECORDS ↵
```

This OPEN command displays information about all files the OPEN command has open; and the CLOSE command closes the file identified by THREE_RECORDS.

```
) OPEN/READ/EOF=** THREE_RECORDS ↵
THREE_RECORDS
```

This command extracts and displays the file's identifier, which is the same as the filename supplied. It opens file THREE_RECORDS and prepares to get information from it.

```
) READ/FILEID=THREE_RECORDS ↵
Record 1 of 3.
) READ/FILEID=THREE_RECORDS ↵
Record 2 of 3.
) READ/FILEID=THREE_RECORDS ↵
Record 3 — and last — of 3.
) READ/FILEID=THREE_RECORDS ↵
**
```

These commands read and display three records from file THREE_RECORDS; the last command reaches the end of file and displays the end–of–file string **.

```
) CLOSE/FILEID=THREE_RECORDS ↵
```

This command closes file THREE_RECORDS.

```
) TYPE THREE_RECORDS ↵
Record 1 of 3.
Record 2 of 3.
Record 3 — and last — of 3.
```

This TYPE command displays the contents of file THREE_RECORDS.

OPEN Example 4

The previous example created a file with a known number of records. It then displayed the records in the file with one READ command for each record. This example consists of a macro, READ_ALL.CLI, that reads all the records of any file you specify and writes them into any other file you specify, regardless of the number of records that are in the input file. Macro READ_ALL.CLI invokes macro COPY_LINE.CLI. Both macros appear next with sufficient COMMENT statements to explain themselves.

Macro READ_ALL.CLI

```
COMMENT This is macro READ_ALL.CLI.
COMMENT
COMMENT The first argument it receives is the filename to open for
COMMENT reading.The second argument it receives is the filename to
COMMENT open for writing. For example, the following command would
COMMENT open file OLD_FILE for reading and  NEW_FILE for writing.
COMMENT
COMMENT    READ_ALL  OLD_FILE  NEW_FILE
COMMENT
push
prompt pop
COMMENT Define the string and value it receives from the CLI when
COMMENT the CLI encounters the end of file during a READ command.
str/name=eof **
COMMENT Open the input file and display its file ID -- FILE_IN.
open/fileid=file_in/read/eof=[!str/name=eof] %1%
COMMENT Open the freshly created output file and display its
COMMENT     file ID -- FILE_OUT.
open/fileid=file_out/write/delete/force %2%
COMMENT Display a blank line after the two file IDs.
WRITE
COMMENT    Read the input file and write each of its lines -- that
COMMENT    is, records -- into the output file until all records
COMMENT    in the input file have been read.
copy_line file_out file_in
COMMENT    The input file is read completely, so display a message
COMMENT    and close all files.
WRITE
WRITE All of the records in file %1% have been processed.
close/all
pop
```

Macro COPY_LINE.CLI

```
COMMENT    This is macro COPY_LINE.CLI.  It reads a line from the
COMMENT    input file that calling macro READ_ALL.CLI opened.
COMMENT    If the line is not the end of the file, COPY_LINE.CLI
COMMENT    writes it into the output file that calling macro
COMMENT    READ_ALL.CLI opened.  If the line is the end of the
COMMENT    file, COPY_LINE.CLI does nothing and ends execution.

read/fileid=file_in/str=line
[!neq,([!str/name=line]),([!str/name=eof])]

COMMENT We are not at the end of the file yet.  So, write the
COMMENT    line into the output file ...

         write/fileid=file_out [!str/name=line]

COMMENT    ... and read the next line from the input file.
         copy_line %1% %2%

[!end]
```

Next is an invocation of READ_ALL.CLI to read the records in file THREE_RECORDS (from Example 2) to the terminal.

```
) READ_ALL THREE_RECORDS @CONSOLE
FILE_IN
FILE_OUT

Record 1 of 3.
Record 2 of 3.
Record 3 — and last — of 3.

All of the records in file THREE_RECORDS have been processed.
```

OPERATOR *Command*

Displays or sets the status of the CLI's ability to load from, label, and dump to labeled diskettes (CLI16 only).

Format

OPERATOR [*ON* / *OFF*]

With the argument, turns CLI operator mode on or off. Without the argument, displays the operator mode status for the current CLI process.

When on, operator mode enables the CLI to write, label, and read diskettes sequentially, in a dump or load operation. This is useful when the material you want to dump to or load from diskette exceeds the capacity of a single diskette. Once on, operator mode remains on until turned off or the current CLI process terminates.

Operator mode must be on if you want to process diskettes by label (rather than simply referring to a diskette unit such as @DPJ10). When you turn it on, operator mode remains on until you turn it off or your CLI terminates.

NOTE: The CLI OPERATOR command is not related to the !OPERATOR pseudomacro or the EXEC OPERATOR command, both of which pertain to the presence or absence of a system operator.

- No templates.
- No argument switches.
- Requirement: *Standard.*
- See also: DUMP, LOAD, LABEL.

Why Use It?

Use the OPERATOR command to work with labeled diskettes. Labeled diskettes are useful if you need to record an amount of data that would exceed the capacity of a single diskette, or if an operation requires a specific diskette.

Labeled diskettes also allow you to write more than one dump file on a diskette. If access a diskette by unit name, you can write only one dump file to it. You must carefully note the volids and filenames, however.

For detailed information on using labeled diskettes, refer to Chapter 6 of this manual or to the manual *Managing AOS/VS and AOS/VS II.*

OPERATOR 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.

/LABEL

Turns automatic diskette labeling on (applies to dumps only). When automatic labeling is on, the CLI will write the specified volid on the label regardless of the existing volid. The CLI bases its labels on the on you specify. For example, if you specify VOL01, the CLI will label the first diskette VOL01; it will label the second diskette VOL02, and so on.

When automatic labeling is off, the CLI displays an error message if the diskette's volid does not match the one specified in your dump command.

Use this switch to create labels on unlabeled diskettes, or to relabel diskettes (for example, if you want to write to a diskette whose retention period has not expired). Operator mode must be off when you give the command

OPERATOR/LABEL ON

OPERATOR Example 1

```
) OPERATOR ↵
OFF
) OPERATOR/LABEL ↵
```

The first command checks the current operator mode. The second turns on operator mode and enables automatic labeling.

OPERATOR Example 2

The following dialog shows the use of a diskette drive to dump a file.

```
) OPERATOR/LABEL ON ↵
) SUPERUSER ON ↵
*) ACL @DPJ10 CHRIS,OWARE ↵
*) DUMP/V @LFD:DUMP01:MYFILE ↵

Please insert a diskette if not already inserted
Unit [@DPJ10] Volume ID [DUMP01] ? [Y] ↵
(verifies files dumped)
.
Please insert next diskette if not already inserted
Unit [@DPJ10] Volume ID [DUMP02] ? [Y] ↵
(verifies files dumped)
.
Please remove the diskette.
```

OPERATOR (continued)

*) ACL @DPJ10 +,WARE ↵
*) SUPERUSER OFF ↵
) OPERATOR OFF ↵

User CHRIS turns on operator mode and automatic labeling. Before using the diskette drive, CHRIS turns on Superuser mode and changes the ACL of the unit to prevent anyone else from trying to use the unit during the dump.

The DUMP command uses @LFD to specify a labeled diskette, identifies the first volume ID (volid) as DUMP01, and names the diskette dump filename (MYFILE). Because Chris enabled automatic labeling, the CLI writes the appropriate volid to each diskette, regardless of any existing label information.

After using the unit, Chris restores the original ACL to allow others access to it.

OPERATOR Example 3

In this example, user Jody dumps the entire directory, :UDD:JODY, to labeled diskettes. First, Jody arranges to have the diskette unit ACL changed to JODY,OWARE. Then Jody issues the next commands.

) DIR :UDD:JODY ↵	(Goes to Jody's user directory.)
) OPERATOR/LABEL ON ↵	(Turns operator mode on and enables automatic labeling.)
) DUMP/V @LFD:JODY01:ALLFILES ↵	(Starts the file dump.)
Please insert a diskette *Unit [@DPJ10] Volume ID [JODY01] [Y]* ↵	(The CLI prompts for a diskette, naming its volid; Jody inserts a diskette and presses NEW LINE.)
...(CLI displays names of files dumped)...	(CLI labels diskette with volid JODY01 and dumps files.)
Please insert next diskette *Unit [@DPJ10] Volume ID [JODY92] [Y]*↵	(CLI prompts for the next diskette, incrementing the volid. Jody removes the diskette, inserts a new one, and presses NEW LINE.)
...(CLI displays names of files dumped)...	(CLI labels diskette and dumps files.) Jody repeats the procedure until the CLI signals the end of the dump by prompting

Please remove the diskette

) OPERATOR OFF ↵ (Turns operator mode off.)

Jody arranges to have the diskette unit ACL restored to +,WARE. Jody writes the filename, the volid, and the date on the paper label for the first diskette. Next, Jody writes the volid and filename on the paper labels for the remaining diskettes.

!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	Incorrect Procedure
) PASSWORD/WRITE=encrypted ↵	) WRITE/L=unencrypted my_password ↵
New CLI password: my_password ↵	) PASSWORD/READ=unencrypted ↵
Confirm password: my_password ↵	
Password accepted.	
) LOCK/FILE=encrypted ↵	) LOCK/FILE=unencrypted ↵
) SUPERUSER ON ↵	) SUPERUSER ON ↵
) *Error: Command is locked*	*Error: Command is locked*
) UNLOCK ↵	) UNLOCK ↵
Password: my_password ↵	*Password:* my_password ↵
	Warning: Password did not match
) BYE ↵	) BYE ↵
AOS/VS II CLI32 Terminating ...	*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.

PATHNAME Example 2

```
) TYPE ERROR_LOG ↵
Warning: File access denied, File ERROR_LOG


) PATHNAME ERROR_LOG ↵
:ERROR_LOG

) SEARCHLIST ↵
:UTIL : :UDD1:TERRY
) FILESTATUS :ERROR_LOG :UDD1:TERRY:ERROR_LOG ↵

Directory :UDD1:TERRY:LISTINGS


  Error: File access denied, File :ERROR_LOG

) SEARCHLIST :UDD:TERRY :UTIL ↵

) TYPE ERROR_LOG ↵
.
(text of ERROR_LOG file in :UDD1:TERRY)
.
```

User TERRY has a file named ERROR_LOG in his initial working directory. When he tries to type this file from another directory, he receives the error message *File access denied.* He then uses the PATHNAME command to discover the pathname to ERROR_LOG and finds another ERROR_LOG file in the root directory — to which he lacks Read access. He checks his search list, which shows the root directory (:) before his initial user directory; and he verifies the existence of the two ERROR_LOG files with the FILESTATUS command.

Finally, TERRY corrects his search list so that the system will check his initial working directory before searching the root.

PATHNAME Example 3

```
) PATHNAME XMAC.CLI ↵
:UTIL:TOOLBOX:XMAC.CLI          (Displays pathname of file XMAC.CLI.)

) DIR :UTIL:TOOLBOX ↵           (Makes TOOLBOX the working directory.)

) MOVE/V :UDD:NAT XMAC.CLI ↵    (Moves copy of file to initial directory
XMAC.CLI                        and verifies the move.)
```

!PATHNAME *Pseudomacro*

Expands to the specified file's full pathname.

Format

[!PATHNAME pathname]

This pseudomacro represents the full pathname (beginning at the root directory) of the specified file.

NOTE: The system cannot expand the argument unless the specified file resides in your working directory or in a directory on your search list. If the system cannot locate the file, the pseudomacro returns a null value.

- No templates.
- No argument switches.
- No macro name switches.
- Requirement: *Standard.*
- See also: PATHNAME, SEARCHLIST.

Why Use It?

Use the !PATHNAME pseudomacro to include a file's full pathname in a CLI command argument. The pseudomacro serves as a shorthand method for referring to a file's pathname. It also lets you use a file's pathname without knowing what it is.

!PATHNAME Example 1

In the following example, suppose the full pathname for the file NIM.PR is :UDD1:SANDY:VS:PROGRAMS:NIM.PR. Further, suppose that you want to create a link (called NIM.PR) to this file. Rather than type the entire pathname, you can use the !PATHNAME pseudomacro when you create the link. Do this by giving the following command.

```
) CREATE/LINK NIM.PR [!PATHNAME NIM.PR] ↵
```

!PATHNAME Example 2

The macro EXPAND.CLI contains the following line:

```
)write The full pathname of %1% is [!pathname %1%].
```

The macro takes a filename as an argument, and then displays the file's full pathname. For example,

```
) EXPAND MYFILE ↵
The full pathname of MYFILE is :UDD:TOM:REPORTS:MYFILE.
```

PAUSE *Command*

Delays the CLI by a specified number of seconds.

Format

PAUSE seconds

This command causes the CLI process to pause the specified amount of time. You can specify a period from 0 through 65535.999 seconds. (You can omit the fractional portion, or specify a maximum of three decimal places in the fraction.)

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

Why Use It?

Use the PAUSE command within a macro if you want the CLI to pause for a certain length of time before executing the next command. You can use this command to allow a person enough time to read a message before the macro overwrites the message with other information.

Also, some CONTROL commands to other processes require a few seconds to complete, although they return immediately to the CLI. In a macro, if you want to ensure that a CONTROL command completes before the macro continues executing, use PAUSE.

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=.

PAUSE Example 1

A system's DOWN.CLI macro contains the following lines:

```
.
COMMENT Disable all consoles.
control @EXEC disable/all
pause 10
.
```

The PAUSE command in this macro delays the next part of system shutdown so that the CONTROL command to the EXEC process can complete.

PAUSE Example 2

Macro PULSE.CLI takes your pulse and records it in freshly created file [!USERNAME].PULSE. The macro follows.

```
comment    This is macro PULSE.CLI.
comment    This macro asks you to take your pulse for 15 seconds.
comment    It multiplies this figure by 4, giving your pulse rate
comment    for one minute.  It then writes the pulse to a log file
comment    and includes the date and time.
push
prompt pop
write    Get ready to begin counting your pulse.
pause 4
write    Start counting now ...
pause 15
write    ... stop counting.
pause 1
string   [!read Type your pulse count and then press NEW LINE: ]
var0     [!umultiply [!string],4]
delete/2=ignore [!username].pulse
create/2=ignore [!username].pulse
string   Your pulse on [!date] at [!time] was [!var0].
write/1=[!username].pulse [!string]
pause 1
write    The logfile recording your pulse is [!username].PULSE.
write    Here is your logfile.
type     [!username].pulse
pop
```

PERFORMANCE *Command*

Displays information about the CLI (CLI16 only).

Format

PERFORMANCE

This command displays the following information about the CLI process:

System calls	The number of calls made since the last PERFORMANCE command, and the total number made by this CLI process.
Shared pages	The number of 2–Kbyte pages of shared memory.
Unshared pages	The current number of 2–Kbyte pages of unshared memory, the maximum possible number of unshared pages, and the greatest number of unshared pages used since this CLI started.
Stack faults	The number of stack faults (the need to grow in 2–Kbyte memory pages) that have occurred since this CLI started.

The information displayed describes CLI activity since the last PERFORMANCE command (or since the beginning of the session if you have not issued a PERFORMANCE command yet).

- No arguments.
- Requirement: *Standard.*

Why Use It?

Use PERFORMANCE to obtain information about CLI system call and memory usage.

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.

PERFORMANCE Example

```
) PERFORMANCE ↵
33 / 1129 System calls
Shared: 18 pages
Unshared: Current 5 pages, Possible 14 pages, Highest 5 pages
3 Stack faults
```

This CLI has made a total of 1129 system calls, 33 of them since the last PERFORMANCE command. The CLI is using 18 pages of shared memory, and 5 out of a possible 14 pages of unshared memory (the highest use so far). Three stack faults have occurred.

PERMANENCE *Command*

Displays or sets the permanence attribute of files.

Format

PERMANENCE pathname [*ON* / *OFF*]

This command either displays whether a file's permanence is on or off, or lets you turn permanence on or off for a file.

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

Why Use It?

Use the PERMANENCE command to protect a file from accidental deletion (by turning permanence on), or to allow a permanent file to be deleted (by turning permanence off).

NOTE: In AOS/VS, permanence does not protect a file from deletion if the directory that contains that file is deleted. In AOS/VS II, the system does not allow a directory that contains a permanent file to be deleted; instead it displays the message *Directory delete error.*

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–mmm–yy:hh:mm:ss), date (dd–mmm–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.

PERMANENCE (continued)

/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–mmm–yy:hh:mm:ss), date (dd–mmm–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 processed.

/SORT — (CLI32 only.) Sorts alphabetically the filenames processed. To see the names, you must also use the /V 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

```
PERMANENCE/TYPE=\CPD #:PROJ.– OFF
```

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

```
PERMANENCE/TRAVERSE=\CPD #:PROJ.– OFF
```

/TYPE=typecode — (CLI32 only.) Specifies one or more types of files to process. Valid values of type codes are in Table 2–8.

/V — Displays the filename with its permanence setting. This switch is useful when you use templates and/or date–time switches.

/VERIFY — (CLI32 only) Same as /V.

PERMANENCE Example 1

```
) PERMANENCE MYFILE ↵
OFF

) PERMANENCE MYFILE ON ↵
) DELETE MYFILE ↵
Warning:  Cannot delete permanent file, File MYFILE
```

The first command checks the permanence setting for MYFILE. The next command turns permanence on for that file. An attempt to delete the file then provokes an error message.

PERMANENCE Example 2

```
) PERMANENCE/V  +  ON ↵
.
(CLI lists filenames on which it turned permanence on)
.
```

This command turns on permanence for every file in the working directory.

```
) PERM/V/AFTER/TLM=[!DATE]  +\+.ED ↵

=REPORT_CURRENT OFF
=CH5_2.DOC.SC       OFF
=A_B.SWITCHES       OFF
```

PERMANENCE Example 3

```
) PERMANENCE/V/AFTER/TLM=[!DATE]  +\+.ED  ON ↵

=REPORT_CURRENT
=CH5_2.DOC.SC
=A_B.SWITCHES
```

The first command displays the names and permanence status of all files in the working directory that were modified today, except those whose names end in .ED. The second command set permanence on for those files. (PERMANENCE with date–time switches works with CLI32 only.)

!PID *Pseudomacro*

Expands to the process ID of the CLI.

Format

[!PID]

This pseudomacro returns the process identification number (PID) of the CLI process. There are no leading zeros in the returned number.

- No arguments.
- No macro name switches.
- Requirement: *Standard.*
- See also: !PIDS, WHO.

Why Use It?

Use the !PID pseudomacro to insert the process ID (PID) of the CLI within a command argument. You may want to do this simply to report information (as shown in Example 1), or to ensure that the current PID is a specific value (as in Example 2).

!PID Example 1

```
) WRITE My username is [!USERNAME] and my PID is [!PID].
```
My username is Ishmael and my PID is 23.

This command displays a message that reports the username and PID of user Ishmael's CLI process.

!PID Example 2

(within a macro)

```
[!ueq, [!pid], 2]
        ...
[!else]
        write Sorry – only PID 2 can perform this operation.
[!end]
```

The conditional pseudomacro !UEQ checks the current process ID to determine if the user is PID 2. If so, the macro executes the statements up to the !ELSE pseudomacro. If the user is not PID 2, the macro displays the message following the !ELSE pseudomacro.

!PIDS *Pseudomacro*

Expands to all process IDs on a host.

Format

[!PIDS *[hostname–or–hostID]*]

This command returns the process ID (PID) of each process running on either the local system or the specified host.

- No templates.
- No argument switches.
- No macro name switches.
- Requirement: *Standard* (for a remote host, you must have a profile on the remote system with the same username and password as on the local system).
- See also: !PID, WHO.

Why Use It?

Use the !PIDS pseudomacro to obtain a list of process IDs that are currently running on a system. You can use this pseudomacro also as an argument to a CLI command (such as RUNTIME or WHO), to obtain information about every process on the system.

!PIDS Example 1

```
) WRITE [!PIDS] ↵
1 2 3 5 7 10 24 25 26 135

) WRITE [!PIDS GALILEO] ↵
GALILEO:1 GALILEO:2 GALILEO:3 GALILEO:6 GALILEO:8 GALILEO:11
```

The first command displays the PIDs currently running on the local system. The second command shows the PIDs running on the host GALILEO.

!PIDS Example 2

The macro WHOS.CLI (supplied with the operating system) contains:

```
host/1=ignore/2=ignore%/%&
 [!nequal,(),(%1%)][!nequal,(%1%),([!hid])]%1%[!end][!end]
runtime%/%&
 [!nequal,(),(%1%)][!nequal,(%1%),([!hid])]%1%:[!end][!end]1
who/2=ignore%/% [!pids]
```

This macro executes the CLI WHO command for each PID on the local host. (If the person running it specifies a hostname as an argument, the macro tries to display the runtime statistics for that host before displaying local PIDs.) The WHO /2=IGNORE switch prevents the display of an error message that could result if a process terminates while the system processes this command.

POP *Command*

Returns to the previous CLI environment level.

Format

POP

This command changes your current environment level by moving to the previous (next lower–numbered) level. For example, if your current level is 3, the POP command returns you to Level 2. Because Level 0 is the base level, you cannot use the POP command in Level 0.

When you leave an environment level via the POP command, the settings in that level are lost; if you later use the PUSH command to re–enter that level, the new environment takes on the current settings.

- No arguments.
- Requirement: *Standard.*
- See also: CURRENT, PUSH, LEVEL.

Why Use It?

Use the POP command to return to the previous environment level. This allows you to restore the environment settings, which you may have changed in a higher level. The following commands let you specify environment settings at each level.

CHARACTERISTICS	LISTFILE	SEARCHLIST
CLASS1	LOGFILE	SQUEEZE
CLASS2	PREFIX (CLI32 only)	STRING
DATAFILE	PRIVILEGE	SUPERPROCESS
DEFACL	SYSTEMMANAGER (CLI32 only)	SUPERUSER
DIRECTORY	PROMPT	TRACE
GROUPLIST (CLI32 only)	SCREENEDIT	VAR0 through VAR9
LEVEL		

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=.

/V	Displays the new environment level number.
/VERIFY	(CLI32 only.) Same as /V.

POP Example 1

```
) LEVEL ↵
Level 3

) POP/V ↵
Level 2
```

The first command displays the current environment level. The second command returns you to the previous level, and verifies the move.

POP Example 2

```
) PUSH ↵
) SEARCHLIST :UDD:COMMON:SPECIAL_UTILITIES  [!SEARCHLIST] ↵
) XEQ MYPROG ↵
) POP ↵
```

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 you to the original environment level, restoring the previous search list.

POP Example 3

The following macro uses the PUSH and POP commands, as in the previous example, to temporarily change the CLI environment for the duration of the macro. In this case, the CHARACTERISTICS command enables page mode before displaying the contents of a file passed to the macro as an argument. The POP command restores the CLI environment to what it was when the macro was called.

```
PUSH; PROMPT POP
CHARACTERISTICS/PM
TYPE %1%
POP
```

POP Example 4

The macro POPBACK.CLI contains the following lines:

```
comment This is macro POPBACK.CLI.
[!ugt, [!level], 0]
        pop
        %0%
[!end]
```

This macro checks the current CLI environment level. If the level number is greater than 0, the macro moves down a level and calls the macro again. When the level number is equal to zero, the macro does nothing.

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 ↵
)
```

PREFIX Example 2

You turn history reporting on in the prefix to view the current command count in the history buffer. In this example, PREFIX was the second command executed after logon.

```
) PREFIX/HISTORY=ON ↵
2!)
```

You execute a command to increment the HISTORY counter in the prefix, then turn the HISTORY counter off in the prefix and restore the default prefix.

```
2!) TIME ↵
12:33:01
3!)
3!)  PREFIX/HISTORY=OFF ↵
)
```

PREVIOUS *Command*

Displays the settings for the previous CLI environment level.

Format

PREVIOUS

This command displays information about the *previous* CLI environment level. The previous level is always the next lower–numbered level (that is, the level you would be in after issuing a POP command). For example, if you are at level 1, the previous level is 0.

- No arguments.
- Requirement: *Standard.*
- See also: CURRENT, POP, PUSH.

Why Use It?

You can use this command to obtain a summary of the CLI environment settings that are in effect in the previous (next lower–numbered) level without using the POP command to return to the previous level.

Commands that pertain to environment settings usually support the /P switch, which lets you refer to a setting in the previous level. You might want to use the PREVIOUS command to check the setting before using the /P 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 switch /STR=.

PREVIOUS Example

```
) LEVEL ↵
1

) PREVIOUS ↵
LEVEL           0
SUPERUSER       OFF
SUPERPROCESS    OFF
...
...CHARACTERISTICS              605X/LPP=24/CPL=80 ...
```

The first command displays the current CLI environment level, which is Level 1. The PREVIOUS command then shows the settings of the previous level, Level 0.

PRIORITY *Command*

Displays or sets the priority of a process.

Format

PRIORITY [*process–ID* | *processname* | *username:processname*] *[new–priority]*

This command lets you either display or set the priority of a process (subject to the restrictions described next). The priority level can range from 1 (the highest) through 255 (the lowest). You can set a process priority to the same or a lower priority.

You can supply a process ID or a process name. If you supply a simple process name, the CLI assumes your username. You can use either process name form, but only if the process was created with the PROCESS command and /NAME=name switch.

- No templates.
- No argument switches.
- Requirement: *Standard* (to display, or to set the priority for the current process or its sons); *Superprocess* (to set the priority for any process).

Why Use It?

Use the PRIORITY command to determine at what priority a process is running. Or, assuming you have the necessary privileges, you can change a process priority, to increase or decrease its share of processing time.

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=.

PRIORITY Example 1

```
) PRIORITY 17 ↵
2

) PRIORITY 17 3 ↵

) PRIORITY 17 ↵
3
```

The first command displays the priority of PID 17. Because PID 17 is a subordinate process of the current CLI, the second command successfully resets PID 17's priority to 3. The last command verifies the new priority.

PRIORITY Example 2

```
) WHO 24 ↵
PID:   24 SULLY        CON5        :CLI.PR

) PRIORITY  SULLY:CON5 ↵
4

) PRIORITY  SULLY:CON5 2 ↵
Error:  Attempt to access process not in hierarchy

) SUPERPROCESS  ON ↵
+) PRIORITY  SULLY:CON5  2 ↵
+) SUPERPROCESS  OFF ↵
) PRIORITY  24 ↵
2
```

In this example, the Superprocess privilege is needed to change the priority of a process that does not belong to the current CLI's process tree.

PRIORITY Example 3

```
) PROCESS/BLOCK/SONS/IOC/PRIORITY=2   :CLI ↵

) PRIORITY ↵
2

) PRIORITY [!PID]  3 ↵
) PRIORITY ↵
3
```

The first command creates a subordinate CLI process and uses the /PRIORITY switch to assign it a priority of 2, as verified by the next command. The third command changes the CLI's priority to 3, which the last command verifies.

PRIVILEGE *Command*

Sets or displays the privilege settings.

Format

PRIVILEGE [*SUPERPROCESS* | *SUPERUSER* | *SYSTEMMANAGER*] [*ON* | *OFF*]

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.

PRIVILEGE Examples

The following example shows someone with System Manager privilege changing the system time to correct for the change from daylight savings time to standard time. (To advance the time, as in the Spring, you should issue this command from PID 2 only, with EXEC shut down, to avoid confusing EXEC as it tracks console connect times.)

```
) TIME ↵
9:30:22
) TIME 8:30:40 ↵
Error: Caller not privileged for this action

) PRIVILEGE SYSTEMMANAGER ON ↵
Sm) TIME 8:31 ↵
Sm) TIME ↵
8:31:11
Sm) PRIVILEGE ↵
Systemmanager
Sm) PRIVILEGE SYSTEMMANAGER OFF ↵
)
```

PROCESS *Command*

Creates a process.

Format

PROCESS pathname *[argument-to-new-process] [...]*

This command creates a new process, which runs as a subordinate process of the CLI. The CLI creates a son process with the program specified by pathname (it first tries to run pathname.PR; if that fails, it tries pathname). You select the type, priority, and privileges of the new process via command switches. Note that if you use the /IOC,/INPUT, /OUTPUT, /LIST, /DATA, or the /STRING switch, you must also select /BLOCK.

If you omit all privilege switches and /DEFAULT, the new process has no privileges. If you use /DEFAULT, then the new process has the same privileges as the creating process. If you omit the /PREEMPTIBLE and /RESIDENT switches, the process is swappable.

The arguments to the new process are placed in the initial IPC message to the new process. The new process can access the arguments through the ?GTMES system call. For information about how a program started from the CLI can access the arguments and switches included in the CLI command line, see call ?GTMES in *AOS/VS, AOS/VS II, and AOS/RT32 System Call Dictionary, ?A through ?Q.*

The list file and data file passed to the son process (with or without the /LIST and /DATA switches) are the CLI's generic list file and data file, @LIST and @DATA. The new process must set these to other files if it wants to use them.

- No templates.
- Accepts argument switches (as appropriate for the program specified by pathname).
- Requirement: *Standard,* except for the following switches:
  - /ACCESSDEVICES requires *Access devices* privilege;
  - /BLOCK (omitting this switch requires *Create without block* privilege;)
  - /CHTYPE requires *Change type* privilege;
  - /CHPRIORITY requires *Change priority* privilege;
  - /CHUSERNAME requires *Change username* privilege.
  - /LOCALITY requires *Change locality* privilege to specified locality.
  - /PREEMPTIBLE requires *Change type* privilege.
  - /RESIDENT requires *Change type* privilege.
  - /SUPERPROCESS requires *Superprocess* privilege;
  - /SUPERUSER requires *Superuser* privilege.

  The /PMGRPRIVILEGES switch also requires special privileges.
- See also: EXECUTE, XEQ.

PROCESS (continued)

Why Use It?

Use the PROCESS command to execute a program and specify detailed, nondefault information about the process, or if you want to create a process that runs concurrently with the current process (without blocking). In many cases, you can use XEQ or EXECUTE instead of PROCESS with satisfactory results.

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=.

/ACCESSDEVICES	Allows the new process to identify and access user devices via system calls ?IDEF, ?DEBL, and ?STMAP.
/BLOCK	Blocks this CLI until the new process terminates. If you omit this switch, the CLI does not block and displays the new process ID. You may omit the switch only if your user profile lets you create a process without blocking. The CHECKTERMS command can display the termination message from the created process when it terminates. You must use /BLOCK if you use the /IOC, /STRING, /INPUT, /OUTPUT, /LIST, or /DATA switches.
/BREAK	Creates a break file if an error trap or fatal termination occurs. The default is no break file.
/BSON	Blocks the son process until explicitly unblocked via the UNBLOCK command or the equivalent system call.
/CALLS=n	Sets the maximum number of concurrent system calls allowed the new process. The default number is the same as that of the creating process.
/CHLOGICALTYPE	Allows the new process to change its logical type (from 16–bit to 32–bit or vice–versa).
/CHPRIORITY	Allows the new process to change its priority.
/CHTYPE	Allows the new process to create any other type of process. Also this switch permits the new process to change its own process type.
/CHUSERNAME	Allows the new process to create a process with a different username than its own.
/CHWSS	Allows the new process to change its working set size.
/CONSOLE	Makes the new process's console the same as that of the creating process. You can also specify a console with /IOC. The default, if you omit both switches, is no console.
/CONSOLE=consolename	Specifies a console for the new process.

PROCESS (continued)

/CPU=s	Limits the CPU time for a new process, where s is a number of seconds in the range 0 through 429,496.
/DACL	Does not pass the default ACL to the new process.
/DATA	Makes the new process's generic @DATA filename the same as that of the creating process. By default, the CLI's @DATA file is set to @DATA (it's not set to a file). You must also specify /BLOCK.
/DATA=pathname	Assigns the generic @DATA file for the new process.
/DEBUG	Starts the new process in the debugger.
/DEFAULT	Gives the new process the same privileges as the creating process; the default is no privileges.
/DIRECTORY	Assigns the initial directory of the creating process to the new process; the default is the working directory of the creating process.
/DIRECTORY=pathname	Assigns an initial directory for the new process.
/DUMP	Includes a dump of the memory image if a break file is created for this process.
/GROUP=groupname	(CLI32 with AOS/VS II only.) Defines a user group that the new process belongs to. This group must be defined in the groups directory, :GROUPS, and you must be allowed to join it. When the new process starts, the groupname will be in its group list. You can use this switch more than once. Groups are further defined in the GROUPLIST command, Chapter 2, and *Managing AOS/VS and AOS/VS II*.
/INPUT	Makes the new process's generic @INPUT filename the same as the creating process's. The default, for an interactive CLI, is your keyboard; for batch, it's the batch input file. You must also specify /BLOCK.
/INPUT=pathname	Assigns the generic @INPUT file for the new process.
/IOC	Assigns the @INPUT, @OUTPUT, and @CONSOLE files of the creating process to the new process. You must also specify /BLOCK.
/IOC=consolename	Assigns consolename as the generic @INPUT, @OUTPUT, and @CONSOLE names for the new process. This works only if EXEC has not enabled the console.
/IPCUSAGE	Allows the new process to issue the primitive Interprocess Communication (IPC) calls.

PROCESS (continued)

/LIST	Makes the new process's generic @LIST file the same as that of the creating process. Running interactively, the default CLI list file (@LIST) is your console. Running in batch, the default CLI list file is the line–printer queue, LPT. You must also specify /BLOCK.
/LIST=pathname	Assigns pathname as the generic @LIST filename for the new process.
/LOCALITY=n	Specifies a user locality for the new process, where n can be from 0 through 15. This switch is meaningful only if your system has nondefault user localities defined (via the PREDITOR utility) and has classes defined and enabled (via the optional CLASP utility).
/MEMORY=pages	Makes pages the maximum memory size of the new process in 2–Kbyte pages. The default is 512 Mbytes. For 16–bit programs, the default is 64 Kbytes.
/NAME=name	Makes name the simple process name for the new process. You can access the process by this name as well as by its PID. If you omit this switch, the system assigns the name.
/NOBLOCKPROC	Allows the new process to create another process without blocking.
/OUTPUT	Assigns the generic @OUTPUT filename of the creating process to the new process. The default is no @OUTPUT. You must also specify /BLOCK.
/OUTPUT=pathname	Assigns the generic @OUTPUT filename for the new process.
/PMGRPRIVILEGES	Allows the new process all the rights of the peripheral manager (PMGR).
/PREEMPTIBLE	Makes the new process pre–emptible; the default is swappable.
/PRIORITY=n	Makes n the new process's priority; the default is the same as the priority of the creating process.
/RESIDENT	Make the new process resident; the default is swappable.
/SONS	A son process may create the same number of processes as the creating process, minus the number already running. If you omit /SONS, /SONS=n, and /UNLIMITEDSONS, the default for creating sons is zero.
/SONS=n	Makes n the maximum number of son processes that the new process can create. If you omit /SONS, /SONS=n, and /UNLIMITEDSONS, the default for creating sons is zero.
/STRING	Stores the program termination IPC message in the current CLI String instead of displaying it. You must also specify /BLOCK.

PROCESS (continued)

/SUPERPROCESS	Allows the new process to enter Superprocess mode.
/SUPERUSER	Allows the new process to enter Superuser mode.
/UNLIMITEDSONS	Allows the new process the option of creating an unlimited number of son processes. If you omit /SONS, /SONS=n, and /UNLIMITEDSONS, the default is zero.
/USERNAME=name	Makes name the new process's username; the default is your username. The default ACL is username,OWARE.
/WSMAX=pagenum	Specifies the maximum number of pages allowed in main memory at one time; the default is dynamically set by the system.
/WSMIN=pagenum	Specifies the minimum number of pages that must be in main memory; the default is dynamically set by the system.

PROCESS Example 1

(within a system's UP.CLI macro)

```
.
process/default/directory=@/name=exec EXEC
.
```

This macro statement starts the EXEC process. The switches assign the EXEC process the same privileges as the calling process, specify the :PER directory (represented by @) as the initial directory, and assign the name EXEC to the process. Assigning the name EXEC lets the system operator access the process by the name EXEC.

PROCESS Example 2

```
) PROCESS/BLOCK/SONS/IOC/PRIORITY=2 :CLI ↵
```

This command creates a subordinate CLI process. The switches block the parent CLI, allow the new process to create the same number of sons (less one), pass the parent CLI's generic files to the new CLI, and assign it the priority of 2. The XEQ command (XEQ :CLI) would do precisely the same thing as this PROCESS command.

PROCESS Example 3

```
) DELETE/2=IGNORE INFILE OUTFILE ↵
) CREATE/I INFILE ↵
)) RP MEMO ↵
))) ↵
) CREATE OUTFILE ↵
) PROCESS/INPUT=INFILE/OUTPUT=OUTFILE :CLI ↵
```

The first CREATE command creates a file named INFILE which contains the macro command RP in it. The second CREATE command creates the output file needed by the PROCESS command that follows. The PROCESS command with its switches creates a CLI process that runs concurrently with its parent, using INFILE as input. The process executes the RP macro using MEMO as its argument. (This works only with the privilege Create without block.) The command also puts the new CLI's output into OUTFILE.

PROCESS Example 4

One application of the PROCESS command is to keep a log of an interactive program's dialog with its user. For example, suppose you have written program FILTER whose purpose is to get the name of an input file, get the name of an output file, and extract some records from the input file into the output file. The logic for extracting these records is unimportant here. You are creating the documentation for this program, so you want to place the program's prompts and your input into a file instead of tediously writing them on paper by hand as the prompts and your responses appear on the screen. Proceed as follows.

Make sure that the dialog file exists as an empty file.

```
) DELETE/2=IGNORE  FILTER.DIALOG ↵
) CREATE  FILTER.DIALOG ↵
) PROCESS/BLOCK/IOC/OUTPUT=FILTER.DIALOG  FILTER ↵
```

The program now executes, sending its prompt messages to file FILTER.DIALOG. No prompt appears on the terminal. But since you know the program you know what questions it will ask. This particular program asks the input and output filenames, then proceeds and terminates. So you type the two filenames:

```
FILTER.INFILE ↵
FILTER.OUTFILE ↵

..(program  runs)...
)
```

Now you can see the dialog by typing the file specified with the /OUTPUT= switch:

```
) TYPE  FILTER.DIALOG ↵
What is the input filename?  FILTER.INFILE

What is the output filename?  FILTER.OUTFILE

Processing input file...

189 records placed into the output file.
End of Job
)
```

Notice that the dialog file contains all of the program's input from @INPUT and output to @OUTPUT.

PROMPT *Command*

Displays or sets the current CLI prompt commands.

Format

PROMPT *[command] [...]*

This command displays the current CLI prompt setting (if you supply no argument). The prompt setting shows what command(s), if any, the CLI executes prior to displaying its prompt.

You can assign a prompt setting by supplying as an argument one or more CLI commands. You can assign up to eight CLI commands for the prompt. You cannot use any command switches or arguments with these commands.

NOTE: If you use the POP command as part of a prompt setting, be sure it is the *last* command in the series; otherwise the CLI will not execute the other commands. As soon as the POP command changes the environment level, the previous level's prompt setting takes effect.

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

Why Use It?

You can use this command if you want the CLI to display information with each prompt. You might, for example, want the name of the working directory and the current time displayed before each prompt. (See Example 1.)

Example 2 shows how you can display the CLI commands currently associated with the prompts, or clear the current PROMPT settings so that no commands are executed.

You can use this command within a macro to ensure that the user returns to the original CLI environment level upon exiting the macro. Often a macro will push one or more levels to avoid altering settings in the original level. (See Example 3.)

To change the prompt character itself [ )], use the PREFIX command.

PROMPT 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=.

/K	Sets the prompt to null. (No arguments are allowed.)
/LEVEL=n	(CLI32 only.) Sets the prompt to that 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, 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 prompt to the one used in the previous CLI environment level. With =*n* (CLI32 only), sets the prompt to the one used in the specified environment level. The *n* specifies the number of levels above the current level (toward 0). No other arguments are allowed.
/PREVIOUS*[=n]*	(CLI32 only.) Same as /P.

PROMPT Example 1

To have the CLI display the current time, date, and working directory with each prompt, and then reset the prompt to null, enter the following commands and observe the results.

```
) PROMPT ↵

) PROMPT TIME DATE DIRECTORY ↵
9:32:16
26-JAN-90
:UDD:ANDREA)
) PROMPT/K ↵
)
```

PROMPT Example 2

The following macro begins by moving down one CLI environment level so that the macro will not affect any settings in the original level. The PROMPT command has the CLI execute a POP command before displaying the CLI prompt, which will happen when the macro finishes execution.

The POP command ensures that the macro returns to the original environment even if a CTRL–C CTRL–A sequence interrupts the macro. Afterwards, the CLI will no longer pop a level with each prompt, because that prompt setting applied to the higher environment level, which is no longer in effect.

```
push
prompt pop
...
```

PRTYPE *Command*

Displays or sets the type of a process.

Format

PRTYPE [*process-ID* | *processname* | *username:processname*] [*PREEMPTIBLE* | *RESIDENT* | *SWAPPABLE*]

This command lets you either display or set the type of a process. A process can be one of the following types:

PREEMPTIBLE	The system can swap the process out of memory only after all swappable processes have been swapped.
RESIDENT	The system cannot swap the process out of memory; the process is resident in memory.
SWAPPABLE	The system can swap the process out of memory as needed. This is the default process type.

You can supply a process ID or a process name. If you supply a simple process name, the CLI assumes your username. You can use either process name form, but only if the process was created with the PROCESS command and /NAME=name switch.

- No templates.
- No argument switches.
- Requirement: *Standard* to display the type of any process or to set the type for a subordinate process; *Superprocess* or the *Change Type* privilege to set the type for any nonsubordinate process.

Why Use It?

Use the PRTYPE command to determine whether a specific process is swappable, pre–emptible, or resident. You can also use this command to change a process type, provided you have the necessary privileges.

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=.

PRTYPE Example 1

```
) PRTYPE ↵
SWAPPABLE
```

This command displays the process type of the current CLI process.

PRTYPE Example 2

```
) PROCESS/PREEMPTIBLE  SMITH:WATCHER ↵
PID:   14

) PRTYPE 14 SWAPPABLE ↵
```

The first command creates a pre–emptible process, which runs as PID 14. The PRTYPE command changes the process type to swappable.

PUSH *Command*

Moves down to the next CLI environment level.

Format

PUSH

This command changes your current environment level by moving up to the next higher–numbered level. If your current level is 2, the PUSH command places you in Level 3, for example.

- No arguments.
- Requirement: *Standard.*
- See also: CURRENT, POP, LEVEL.

Why Use It?

Use the PUSH command to move up to the next environment level. Doing so allows you to alter the environment settings, perhaps to execute a macro, and then use the POP command to return to the previous level and restore the original environment.

The following commands let you specify environment settings at each level:

CHARACTERISTICS	LISTFILE	SEARCHLIST
CLASS1	LOGFILE	SQUEEZE
CLASS2	PREFIX (CLI32 only)	STRING
DATAFILE	PRIVILEGE	SUPERPROCESS
DEFACL	SYSTEMMANAGER (CLI32 only)	SUPERUSER
DIRECTORY	PROMPT	TRACE
GROUPLIST (CLI32 only)	SCREENEDIT	VAR0 through VAR9
LEVEL		

You can establish different environment settings in each level.

Using PUSH within a macro enables you to temporarily change the CLI environment for the duration of the macro (to change the working directory, for example). At completion, the macro can restore the previous CLI environment by issuing a POP command. (See Example 3.)

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=.

/V	Displays the new environment level.
/VERIFY	(CLI32 only.) Same as /V.

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.

QBATCH (continued)

/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.

QBATCH Example 1

```
) QBATCH XEQ MYPROGRAM ↵
Queued, Sequence number = 51, Qpriority = 127

) QDISPLAY/QUEUE=BATCH_INPUT ↵

BATCH_INPUT   BATCH  Open
  51 D  LEE    :UDD:LEE:TEST_PROGRAMS:MYPROGRAM.PR

Flags explanation:
D = /DELETE
* = Active
```

The QBATCH command submits the job XEQ MYPROGRAM to the batch queue. The QDISPLAY command shows the job in the BATCH_INPUT queue.

QBATCH Example 2

(within a macro)

```
qbatch/m
directory :udd:lee:tests
searchlist :udd:lee :util
xeq test %1%
```

This series of statements within a macro creates a batch job. The XEQ command is completed with argument information that the user supplies when calling the macro.

QBATCH Example 3

```
) QBATCH/AFTER=19-JUN-90:17/JOBNAME=RPJOB RP FILE1 ↵
QUEUED, SEQ=9232, QPRI=127      (Queues a batch job with jobname RPJOB to
                                 execute after 5 p.m. on June 19, 1990.)

) QDISPLAY/V/QUEUE=BATCH_INPUT ↵

BATCH_INPUT  BATCH  Open        (Displays batch queue data.)

SEQ#  PRI  Time       Max Time  FLGS  Username Jobname  Pathname
           24-MAY-89
9232  127  15:43:40   0:01:00   DA    MARLL    RPJOB   :UDD1:MARLL:CLI
                                                       :?18.CLI.00001.JOB
After 19-JUN-89 17:00:00

Flags explanation:
D = /DELETE
A = Unexpired /AFTER

) QCANCEL RPJOB ↵               (Cancels batch job named RPJOB.)
```

QCANCEL *Command*

Cancels an entry in a queue.

Format

```
QCANCEL { sequence-number } [ ... ]
        { jobname       }
        { ,             }
```

This command cancels a batch or print job submitted to a queue with the QBATCH, QFTA, QPLOT, QPRINT, QSNA, QSUBMIT commands. The job can be active (actually printing on the printer) or inactive (waiting in the queue). The request remains in the queue, with status letters beside it, until an EXEC cooperative process cancels the job. You can cancel a request by job name only if it was queued by the QBATCH command with the /JOBNAME=name switch.

To cancel all your batch jobs that have a null job name, use a comma to indicate a null argument.

If you want to change the characteristics of a job without canceling it, use the QMODIFY command.

- No templates.
- No argument switches.
- Requirement: *Standard*.
- See also: QDISPLAY, QMODIFY.

Why Use It?

Use the QCANCEL command to cancel a request to process a job. For example, you may have used the QPRINT command to print several copies of a file, but then realized the file you specified is not the most up-to-date version. You can use QCANCEL to discard that job, and then issue another QPRINT command to print the correct 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 switch /STR=.

QCANCEL Example 1

```
) QPRINT/COPIES=10 MYFILE ↵
Queued, Sequence number = 45, Qpriority = 127

) QDISPLAY/QUEUE=LPT ↵
LPT    PRINT    OPEN
```

(Several jobs are listed.)

```
) QCANCEL 45 ↵
```

The first command requests 10 printed copies of MYFILE. The second command checks the line printer queue, revealing several entries already in the queue. The last command cancels the print request, referring to the job by its sequence number.

QCANCEL Example 2

```
) QCANCEL , ↵
```

This command cancels all batch queue requests submitted by the current username.

QCANCEL Example 3

```
) QBATCH/AFTER=19-JUN-89:17:00/JOBNAME=RPJOB RP FILE1 ↵
QUEUED, SEQ=9232, QPRI=127        (Queues a batch job with jobname RPJOB
                                   to execute after 5 p.m. on June 19, 1989.)

) QDISPLAY/V/QUEUE=BATCH_INPUT ↵

BATCH_INPUT   BATCH  Open          (Displays batch queue data.)

SEQ#  PRI  Time        Max Time  FLGS  Username  Jobname  Pathname
           24-MAY-89
9232  127  15:43:40   0:01:00   DA    MARLL     RPJOB    :UDD1:MARLL:CLI
                                                          :?18.CLI.00001.JOB
After 19-JUN-89 17:00:00

Flags explanation:
D = /DELETE
A = Unexpired /AFTER

) QCANCEL RPJOB ↵                 (Cancels batch job named RPJOB.)
```

QDISPLAY *Command*

Displays queue information.

Format

QDISPLAY *[hostname]*

This command displays information about the queues on your system. If your system is running the XODIAC/XTS Resource Management Agent (RMA), you can supply a hostname argument to obtain information about the queues on that host.

The output from this command shows the queue name, queue type, and status for each queue. See the examples for the output format.

QDISPLAY works only when EXEC is running.

QDISPLAY can tell you if the printer is available, how many jobs precede yours, the sequence number of your request or batch job and the status of your request or job. (See QBATCH, QCANCEL, QFTA, QPLOT, QPRINT, QSNA, and QSUBMIT commands.)

The display uses the following codes to describe individual jobs.

Code	Meaning
*	The job is active (being printed, plotted, and so on).
+	The job is awaiting a response from the system operator.
A	An unexpired /AFTER switch is in effect.
B	A /BINARY switch is in effect.
C	The job was canceled (via QCANCEL) by its owner. A /DELETE switch is in effect.
E	The job is being held by the system operator.
F	The job was canceled by the system operator.
G	A /NORESTART switch is in effect.
H	The job is being held (via /HOLD or QHOLD) by its owner.
N	A /NOTIFY switch is in effect.
O	An /OPERATOR switch is in effect.
R	The system restarted the job following a system failure.
S	The job was submitted by a Superuser.

- No templates.
- No argument switches.
- Requirement: *Standard*.
- See also: QCANCEL, QFTA, QMODIFY, QPLOT, QPRINT, QSNA, QSUBMIT.

Why Use It?

Use QDISPLAY to determine the status of one or more queues. Before submitting a job to a queue, you can use QDISPLAY to see if there is a backlog of jobs already in the queue. Or after you submit a queue request, you can use QDISPLAY to see where your job stands in the queue.

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=.

/QUEUE=queuename — Displays entries for the specified queue only, such as LPT. You may use this switch more than once to select several queues. See your system operator for your system's queue names. (You cannot abbreviate a queue name.)

/SUMMARY — Displays queue names, types, open/closed status, and a count of requests in each queue.

/TYPE=type — Displays only queues of the specified type, which can be a unique abbreviation of any of the following:

BATCH	PLOT
FTA	PRINT
HAMLET	SNA
MOUNT	

You can use /TYPE= more than once to display several queue types.

/V — Displays column headings and more detailed information for each queue that has entries. (This switch is ignored if you use /SUMMARY.)

/VERBOSE — (CLI32 only.) Same as /V.

QDISPLAY Example 1

```
) QDISPLAY ↵
BATCH_INPUT     BATCH    Open
BATCH_OUTPUT    PRINT    Open
BATCH_LIST      PRINT    Open
MOUNTQ          MOUNT    Open
LPT             PRINT    Open
FTQ             FTA      Open
LQP             PRINT    Open
```

This command shows the status of all queues on the system. The first column lists the queue names; the second column shows the queue type, and the third column reports the current status of the queue. Queued jobs appear below the appropriate queue name.

QDISPLAY Example 2

```
) QDISPLAY/QUEUE=LPT ↵
LPT             PRINT    Open
```

This command lists the entries for a specific queue, LPT, which is associated with a line printer.

QDISPLAY Example 3

```
) QBATCH/NOTIFY XEQ MYPROG ↵
Queued, Seq = 16466, Qpriority = 127
) QDISPLAY/QUEUE=BATCH_INPUT↵

BATCH_INPUT    BATCH  Open
*16466 DN  TOMR   :UDD1:TOMR:VS:?42.CLI.00001.JOB

Flags explanation:
S = Queued by superuser
D = /DELETE
N = /NOTIFY
A = Unexpired /AFTER
* = Active

) QCANCEL , ↵
```

The previous example shows someone posting a batch job via QBATCH, then displaying job status with QDISPLAY/QUEUE=, and finally cancelling the job with QCANCEL. The status display indicates that the job is active (*), the batch input file will be deleted after processing and that the person wants to be notified when the job is done (flags DN), that the username is TOMR; and that the batch input pathname is :UDD1:TOMR:VS:?42.CL I.00001.JOB. The pathname indicates the directory from which the job was started.

QFTA *Command*

Submits an entry to the File Transfer Agent (FTA) queue.

Formats

```
QFTA { /DESTINATION=:NET:hostname:pathname pathname
     { /DESTINATION=pathname :NET:hostname:pathname
     { /DESTINATION=pathname  pathname
```

This command submits a specified file to the File Transfer Agent (FTA) queue. You can check FTA queue status with the command QDISPLAY/TYPE=FTA.

NOTE: The command does not actually transfer the file, but only provides the file pathname to the FTA queue. Do not delete or modify the file until FTA has transferred a copy of it.

FTA puts a log file containing a record of the transaction into :UDD:username (your username) in the form FTA.OUTPUT.n, where n stands for the job's sequence number. You can type this file to discover whether your request completed normally. You may want to delete these FTA.OUTPUT.+ files periodically to reclaim disk space.

You can give the destination file a name that is either the same as or different from the source file. If the destination file in pathname already exists, you will receive a *File already exists* error message. If you really want to overwrite the file, use the /RECENT or /DDELETE switch.

The next sections describe each format separately.

QFTA Format 1

QFTA/DESTINATION=:NET:hostname:pathname pathname

Use this form to transfer a local (source) file to a remote (destination) system. The hostname is the remote system's name; pathname is the full pathname to the directory in which you want to place the file; and pathname is the name of the file you are transferring. You must have the same username/password pair on both systems.

QFTA Format 2

QFTA/DESTINATION=pathname :NET:hostname:pathname

Use this form to transfer a remote file to a your local system. You must have the same username/password pair on both systems.

QFTA Format 3

QFTA/DESTINATION=pathname pathname

Use this form to transfer a file on your local system. Use this format (instead of theMOVE, RENAME, or COPY commands) when you want to transfer a very large file, using the /AFTER= switch on the command. The destination pathname must include the full pathname (can start at the root) to the directory in which you want to place the file; and the pathname is the name of the file you are transferring.

QFTA (continued)

- Accepts templates.
- No argument switches.
- Requirement: *Standard,* but to transfer a file between a local and remote system, you must have the same username/password combination on each.
- See also: MOVE/FTA, QCANCEL, QDISPLAY, QMODIFY.

Why Use It?

Use the QFTA command to transfer a file between your local computer system and a remote system. Although you can perform the same operation by using the MOVE or COPY command with the /FTA switch, the QFTA command lets you submit the transfer request to a queue so that you do not have to wait until the transfer is complete. (FTA can negotiate some transfers across intermediate hosts that MOVE alone cannot.)

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=date:time
: Processes this request after the specified date and time. Use the format dd–mon–yy for date; use hh:mm:ss for time; if you specify both, separate them with a colon — for example, 15–MAY–90:9:00. Or use the format +hh:mm:ss alone to specify a delay from the current time.

/APPEN
: Appends the source file to the destination file if the destination file already exists.

/CHECKPOINT
: Restarts the transfer from the most recent checkpoint if a network error occurs, if either the remote or local system panics, or if the local EXEC terminates during the transfer. This switch and /NORESTART are mutually exclusive.

/COMPRESS*[=n]*
: Compresses data during transmission. You can specify a data compression algorithm n; currently the only available algorithm is 0. If you omit the switch altogether, the system does not compress the data.

/DDELETE
: Deletes any existing destination file with the same name as the source file before transferring the file.

/DESTINATION=pathname
: Required switch. Specifies the destination pathname. You must specify a full pathname. The pathname can be remote or local. If remote, it must start with :NET:hostname. If local, it can start with the root (:). In either case, the destination pathname must end with the filename you want to give to the transferred file.

QFTA (continued)

/HOLD	Holds this job in the queue. (You can later use the QUNHOLD command to release the job.)
/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. This switch and /CHECKPOINT are mutually exclusive.
/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.
/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. The queue type must be FTA.
/RECENT	Processes the request only if the source file is more recent than the destination file.
/RMODE	Uses record mode transfer; the default is block mode.
/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.
/SDELETE	Deletes the source file after the transfer is completed.
/STRING	(CLI32 only.) Same as /S.
/V	Displays the pathname of the queued file.
/VERIFY	(CLI32 only.) Same as /V.

QFTA Example 1

The following command asks FTA to transfer a local file to a remote host.

```
) QFTA/V/NOTIFY/RECENT/DESTINATION=:NET:TITAN:UDD:TERRY:FILE1  FILE1
:UDD:TERRY:REPORTS:FILE1Queued, Sequence Number = 32, Qpriority = 127

) QDISPLAY/TYPE=FTA ↵

FTQ       FTA       Open
 *22  A   ROBIN     :UDD:ROBIN:PROJECTS:NESTING
 32       TERRY     :UDD:TERRY:REPORTS:FILE1
```

Flags explanation:
A = Unexpired /AFTER

.
(time passes)
.

From PID 3: EXEC ...

```
) TYPE   :UDD:TERRY:FTA.OUTPUT.32 ↵
```

.
(status information)
.

This command asks FTA to transfer a file to remote host TITAN. The /NOTIFY switch asks for notification when the request has completed; the /V switch displays the name of the queued file. The /RECENT switch tells FTA to delete and replace the destination file only if the source file is newer (has been modified more recently). The destination pathname is fully qualified with the :NET directory name and the :TITAN hostname.

The QDISPLAY command examines the FTA queue, which shows TERRY's request next in line for processing. When the request completes, EXEC notifies TERRY. TERRY then types the FTA log file (format FTA.OUTPUT.n, n is the job sequence number) to see whether the job completed normally.

QFTA Example 2

The next command requests the transfer of a remote source file to a local directory.

```
) QFTA/DESTINATION=:UDD:TR:SOURCES:18_MAY_90_PARU.SR/V/NOTIFY &↵
&)  :NET:HOST2:UDD:TR:SOURCES:PARU.SR ↵
Queued, Sequence number = 1665, Qpriority = 127
```

This command places a request on the FTA queue to transfer a copy of the remote file PARU.SR (located on HOST2) to directory :UDD:TR on the local system. The command specifies the full pathname to the destination file, ending with a filename that is different from the source file's name. The command argument also specifies the full pathname to the remote source file; this is always required.

QHOLD *Command*

Holds an entry in its queue.

Format

QHOLD { sequence–number | jobname | , } *[...]*

This command lets you temporarily suspend the processing of a queue entry, subject to the restrictions described in the next section.

You can hold all batch jobs that share a job name by supplying that job name as an argument to the command. To hold all batch jobs under your username that do not have a job name, enter a comma as the argument. You can assign a job name for a batch job by using the /JOBNAME switch in the QBATCH command.

The sequence number is the number that the system displays when it receives a QBATCH or QSUBMIT command. To release a job, use the QUNHOLD command; to cancel one, use QCANCEL.

You can use this command only with a job that you have submitted and that is not yet active.

- No templates.
- No argument switches.
- Requirement: *Standard.*
- See also: QCANCEL, QDISPLAY, QMODIFY, QUNHOLD (to release a held job).

Why Use It?

Use the QHOLD command to temporarily hold a job in its queue. You might want to do this to delay processing, perhaps to verify related information before continuing with the request.

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=.

QHOLD Example 1

```
) QDISPLAY/QUEUE=BATCH_INPUT ↵

BATCH_INPUT    BATCH  Open
  730   DA  TERRY   :REV08:CHANGES:?21.CLI.00001.JOB

Flags explanation:
D = /DELETE
A = Unexpired /AFTER
* = Active

) QHOLD 730 ↵

) QDISPLAY/QUEUE=BATCH_INPUT ↵

BATCH_INPUT    BATCH  Open
  730 HDA  TERRY   :REV08:CHANGES:?21.CLI.00001.JOB

Flags explanation:
H = Held by user
D = /DELETE
A = Unexpired /AFTER
* = Active
```

The first command displays the jobs in the BATCH_INPUT queue. User TERRY decides to hold job 730 in the queue. A second QDISPLAY command shows job 730 held in its queue.

QHOLD Example 2

To submit an entry on the batch queue, giving it a job name, you could type

```
) QBATCH/JOBNAME=MYPROG XEQ MYPROG FILE1 ↵
```

This command posts a request with the job name MYPROG on the batch queue. Now type

```
) QHOLD MYPROG ↵
```

This command holds (suspends the processing request for the job) MYPROG until you release it with the QUNHOLD command.

QMODIFY *Command*

Changes information about a job currently in either a batch, plot, or print queue.

Format

QMODIFY/switch[/*switch ...]* sequence–number *[...]*

This command changes the queue parameters of a job that you previously submitted with QBATCH, QPLOT, QPRINT, or QSUBMIT. The command alters the parameters of a job you submitted with QSUBMIT only if the job exists in a batch–, plot–, or print–type queue.

To change a queue parameter, specify the command switch that defines the parameter. The switch you select must be valid for the job type, and must not conflict with any switch that will remain in effect. When you change one or more queue parameters, the rest of the parameters for that job remain the same. To modify a queue parameter of a job on a public queue, include the /QUEUE=queuename switch.

Restrictions

You cannot change the parameters of a your job after it becomes active, or of any job submitted by another user.

- No templates.
- No argument switches.
- Requirement: *Standard*.
- See also: QCANCEL, QDISPLAY, QHOLD.

Why Use It?

Use the QMODIFY command to change a previously submitted job. This command lets you change the job without canceling and then resubmitting 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 switch /STR=.

/8BIT — (Print jobs) Uses all eight bits to determine a character, thus overriding the system default use of only seven bits. This switch isn't necessary if your system is configured to interpret all eight bits. See the explanation of /8BIT under QPRINT.

QMODIFY (continued)

/AFTER=date:time — Processes this request after the specified date and time. Use the format dd–mon–yy:hh:mm:ss for the date and time. Or use the format +hh:mm:ss alone to specify a delay from the current time. For example, /AFTER=+4 (a legal abbreviation of /AFTER=+04:00:00) delays processing for at least four hours after you issue the QMODIFY request. To let a job run immediately, use /AFTER=0.

/BEGIN=n — (Print jobs) Starts printing the file at page n. If you also use the /END= switch, the starting page number must be less than or equal to the ending page number. If you omit this switch, printing begins at the first page.

/BINARY — (Print jobs) Prints in binary mode. This switch is valid only for devices that have binary mode enabled. Check with your operator for available local binary devices. To remove this switch, use /NOBINARY.

/COPIES=n — (Print and plot jobs) Produces n copies of the file. If you omit this switch or do not specify n, one copy is printed or plotted.

/CPU=time — (Batch jobs) 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 — such as one an operator set — and you specify a limit that exceeds that value, the system will not process this request.

/DELETE — (Print and plot jobs) Deletes the file(s) after processing. To remove this switch, use /NODELETE.

/DESTINATION=string — (Batch and print jobs) Prints the specified string in block letters at the top of any header or trailer pages. If you omit this switch, your username is printed.

/END=n — (Print jobs) Stops printing the file at page n. If you also use /BEGIN=, the ending page number must not be less than the starting page number. If you omit this switch or do not specify a page number, printing ends at the last page.

/FOLDLONGLINES — (Print jobs) Continues long lines on the next line of the listing rather than truncating them. To remove this switch, use /NOFOLDLONGLINES.

QMODIFY (continued)

/FORMS=form–filename
(Print and plot jobs) Prints on the specified type of form. Check with your operator for the available form types, such as mailing labels. If you omit this switch or do not provide a type value, standard forms are used.

/HOLD
(All jobs) Holds the entry until you explicitly release it with QUNHOLD or another QMODIFY command. To remove this switch, use /NOHOLD.

/JOBNAME=name
(Batch jobs) 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.

/MAPPER=mapper–filename
(Print jobs) Uses the specified mapper file to translate a character that is not printable into a printable equivalent. See the explanation of /MAPPER under QPRINT.

/NOBINARY
(Print jobs) Does not print the file in binary mode.

/NODELETE
(Print and plot jobs) Does not delete the pathname after processing.

/NOFOLDLONGLINES
(Print jobs) Truncates long lines rather than continuing them on the next line of the listing.

/NOHOLD
(All jobs) Does not hold the entry.

/NONOTIFY
(All jobs) Does not have the system send a message to your terminal upon completion of the queue request.

/NOOPERATOR
(All jobs) Runs the job regardless of whether or not an operator is present.

/NORESTART
(All jobs) 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
(All jobs) Sends a message to your terminal when this job is completed. By default, no message is sent. To remove this switch, use /NOTIFY.

/NOTITLES
(Print jobs) Does not print title lines on each page.

/OPERATOR
(All jobs) Runs the job only if a system operator is on duty. Use this switch when submitting a batch job that contains a MOUNT request. To remove this switch, use /NOOPERATOR.

/PAGES*[=n]*

(Print jobs) Does not print more than n pages. This switch applies only if the operator has specified a page limit for the queue. If the operator has specified a page limit and you use /PAGES without the value n, the system estimates the number of pages in your job as follows:

pages = (bytes–in–file)/1000 + 4

If the operator has set a page limit and the value you specify exceeds that limit, the job will not be processed.

/QLIST=pathname

(Batch jobs) Sets the generic list file of the batch process to this pathname (which may not be a queue name). If you omit this switch or do not supply a pathname, the system creates a temporary list file for the process. Its name is username.LIST.sequence_number. If this temporary file contains text at the completion of the process, the system prints the file.

/QOUTPUT=pathname

(Batch jobs) Sets the output file of the batch process to this pathname (which may not be a queue name).

If you omit this switch or do not specify a pathname, the system creates a temporary output file for the process. If the temporary file contains text when the process completes, the system prints the file.

/QPRIORITY=n

(All jobs) 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

/RESTART

(All jobs) If the system fails during processing, this switch starts the job over again when the system comes up. To remove this switch, use /NORESTART.

/TITLES

(Print jobs) Prints a title line on each page. The title line includes the file's pathname, date and time last modified, and page number. If you omit this switch, the system prints no titles. To remove this switch, use /NOTITLES.

QMODIFY Example 1

```
) QPRINT/AFTER=21:30  MYFILE ↵
:UDD:TOM:MYFILE Queued, SEQ=512, QPRI=127

) QMODIFY/COPIES=4/AFTER=[!TIME]  512 ↵
```

This command changes job 512 to request four printed copies and no time delay. The !TIME pseudomacro represents the current time.

QMODIFY Example 2

```
) QPRINT/AFTER=[!DATE]:19:00   MYFILE ↵
Queued, Sequence number = 489, Qpriority = 127

) QMODIFY/COPIES=4/AFTER=[!DATE]:[!TIME]  489 ↵
```

The QPRINT command tells the system to print file MYFILE after 7:00 p.m. today. The QMODIFY command tells the system to change parameters: to print the file immediately (/AFTER with [!DATE]:[!TIME]), and to print four copies.

QMODIFY Example 3

```
) QBATCH XEQ  MASM SYS SYS1 SYS2 ↵
Queued, Sequence number = 41, Qpriority = 127

) QMODIFY/CPU=00:2:00/JOBNAME=SOURCES/NOTIFY  41 ↵
```

The QBATCH command posts a batch job to run the macroassembler. The QMODIFY command changes the original parameters: it limits CPU usage to 2 minutes, assigns the job name SOURCES, and requests notification at the terminal after the job completes.

QPLOT *Command*

Submits a job to a plotter queue.

Format

QPLOT pathname *[...]*

This command places an entry on the digital plotter queue.

NOTE: The command does not actually plot the specified file, but only provides the file's pathname to the queue. Do not modify or delete the file until you are sure it has been plotted.

The system always plots data exactly as it appears in the file. EXEC does not record billing parameters.

Generally, do not try to plot a file if an entry already exists in the plot queue since the plotter may draw the new plot on the previous one. You can check the queue with the QDISPLAY command.

- Accepts templates.
- No argument switches.
- Requirement: *Standard.*
- See also: QCANCEL, QDISPLAY, QHOLD, QMODIFY.

Why Use It?

Use the QPLOT command to plot a file on a digital plotter.

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=date:time
: Processes this request after the specified date and time. Use the format dd–mon–yy for date; use hh:mm:ss for time; if you specify both, separate them with a colon. Or use the format +hh:mm:ss alone to specify a delay from the current time.

/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–mmm–yy:hh:mm:ss), date (dd–mmm–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.

QPLOT (continued)

/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–mmm–yy:hh:mm:ss), date (dd–mmm–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.

/COPIES=n — Produces n copies of the file. You can print as many as 24 copies; the default number is 1.

/COUNT — (CLI32 only.) Counts the number of files this command processes.

/DELETE — Deletes the files after plotting them.

/FORMS=type — Plots the job using the specified forms. Check with your operator for the types of forms that are available, such as mailing labels. If you omit this switch, the system uses the standard forms. (See /OPERATOR also.)

/HOLD — Holds this job in the queue. You can later use the QUNHOLD command to release the job.

/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. Use this switch when your job needs operator attention (for example, to remove a plot after it is done).

/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 one specified in your user profile. If you omit this switch, the system assigns a priority based on this formula where m represents the priority specified in your user profile:

$n = (m + 255) / 2$

/QUEUE=queuename — Submits the job to the specified queue instead of to the default queue. The queue type must be PLOT.

/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.

QPLOT (continued)

/SORT — (CLI32 only.) Sorts alphabetically the filenames this command processes.

/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

```
QPLOT/TYPE=\CPD #:PROJ.-
```

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

```
QPLOT/TRAVERSE=\CPD #:PROJ.-
```

/TYPE=typecode — (CLI32 only.) Specifies one or more types of files to process. Valid values of type codes are in Table 2–8.

/V — Displays the names of the queued files. This switch is useful if you used a template.

/VERIFY — (CLI32 only.) Same as /V.

QPLOT Example

```
) QPLOT FILE1 ↵
Queued, SEQ=651, QPRI=127
```

This command queues FILE1 to a digital plotter output queue.

QPRINT *Command*

Submits a job to a print queue.

Format

QPRINT pathname *[...]*

This command places an entry in the print queue. You can determine the status of entries in the print queue with the QDISPLAY comand, and you can cancel your own entries with the QCANCEL command.

NOTE: Depending on other jobs, the system may not print the file immediately. Do not modify or delete the file until you are sure it has been printed.

- Accepts templates.
- No argument switches.
- Requirement: *Standard.*
- See also: QCANCEL, QDISPLAY, QHOLD, QMODIFY.

Why Use It?

Use the QPRINT command to print a file on a printing device (such as a line printer or laser printer).

Using a Remote Queue

If you want to submit a job to a remote queue (to use a printer attached to a remote system, for example), use the /QUEUE= switch and specify the remote queue in this format

/QUEUE=hostname:queuename

You must use a full network pathname for a file that you submit to a remote queue unless the file resides on the same host as the queue.

You can use a remote queue only if you have a profile on the remote system with the same username/password combination as your profile on the local system. Also, both systems must have the same password for the operator (OP) process. The /NOTIFY switch does not work with a remote queue.

NOTE: You should not submit large jobs to a remote queue; a failure in the network during processing of the job can cause a problem with the remote site's XLPT program, requiring its queue to be purged.

Commands to the XLPT Printer Process

Within a text file, you can specify commands to the printer manager process, XLPT. Such a command might specify the pathname of another file you want printed, for example.

You specify XLPT commands as command numbers that you enclose with one byte that has the ASCII value 377. You can produce a file that contains the byte, and then insert that file in a text file as needed. Create the file with CLI32, using the WRITE/NONEWLINE switch. If you are runing CLI16, execute :CLI32 for this one operation. For example, to create a file named 377_BYTE with just a 377 in one byte, type

```
) WRITE/NONEWLINE/L=377_BYTE  [!ASCII 377] ↵
```

The format of commands to XLPT is

```
<377>[commandnumber[;argument–bytelength;argument]]<377>
```

where commandnumber is one of the following numbers.

0 Tells XLPT to print a single <377>. This reproduces pre–AOS/VS Revision 7.62 functionality, where you inserted two 377s to instruct XLPT to pass one 377 to the printer.

1 Tells XLPT to include a file. As argument, include the file's full pathname from the root directory. As with any file to read, you must have read access to the file. As argument–bytelength, specify the number of characters in that pathname. The pathname can be a link filename; you can delete and recreate thelink as needed to specify different text files.

2 Tells XLPT to include a file, then delete the included file after printing it. As argument–bytelength, specify the number of characters in that pathname. As argument, include the file's full pathname from the root directory. As with any file to read, you must have read access to the file. The pathname can be a link filename; you can delete and recreate the link as needed to specify different text files.

For example, assume MYFILE contains the text

```
Introduction:

<377>1;17;:UDD:CHRIS:MYFILE<377>

End of Document
```

If you type QPRINT MYFILE, the XLPT process will print Introduction:, followed by two NEW LINE characters and the contents of file :UDD:CHRIS:MYFILE, followed by a NEW LINE character and End of Document.

When XLPT prints an include file, the new file's text begins where the including command begins, without a NEW LINE character or other break in text. The entire file will be included until an end of file is encountered. This means any files that it includes will be printed also. You can nest include files up to a level of seven files (eight if you count the main file).

There are many applications for the include file facility. You can change fonts in the middle of binary print jobs by including a font file at the point desired. Or you can print multiple files without intervening header pages.

If XLPT finds an error in a defined command, or if a file system error occurs, XLPT will print an error message that includes the name of the offending file and flush the job. If XLPT doesn't recognize the command, it will print the entire command text (omitting the first <377>) and continue printing the 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 switch /STR=.

/7BIT — Tells the printer process to use only the rightmost 7 bits when it prints characters. Unless the printer was started, via the EXEC utility, with the /8BIT switch, 7 bits is the default value.

/8BIT — Tells the printer process that all 8 bits identify each character. The default system generation setting tells the printer process that the eighth bit is not part of the character. The /8BIT switch is useful with printers configured for 8–bit characters. For example, the stored value octal 250 prints as a left parenthesis, (, in 7–bit mode, but will print as a British pound symbol (similar to a handwritten letter L) in 8–bit mode.

The EXEC program's START command also has an /8BIT switch, which makes 8–bit mode the standard for a printer queue. If the printer queue was started with this /8BIT switch, you don't need QPRINT's /8BIT switch to print files with 8–bit characters.

If the printer is on an asynchronous line, you should make sure that VSGEN or the CHARACTERISTICS command has configured the line for 8–bit characters.

/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. Or use the format +hh:mm:ss alone to specify a delay from the current time.

QPRINT (continued)

/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–mmm–yy:hh:mm:ss), date (dd–mmm–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.

/BEGIN=n

Starts printing the file at page n. If you also use the /END= switch, the starting page number must be less than or equal to the ending page number. If you omit this switch, printing begins at the first page.

/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–mmm–yy:hh:mm:ss), date (dd–mmm–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.

/BINARY

Prints in binary mode. Tells the printer process to send all characters, except characters with the value octal 377, to the printer.

The value octal 377 has special meaning to the printer process. The process treats all characters between octal 377s as commands as explained above in the section "Commands to the XLPT Printer Process." You can insert the octal 377s and commands in text using the form

<377>command<377>

The process will discard the surrounding octal 377s and interpret the command as a printer command.

For this switch to work, the operator must have enabled binary mode on the target printer. This is the printer served by the queue that you specify. The operator can enable binary mode with the EXEC utility program's command BINARY. See also the explanation of the /PASSTHRU switch below.

To pass an octal <377>, the file must contain a 0 folllowed by <377>, as in 0<377>.

/COPIES=n

Prints n copies of the file. The default (if you omit this switch) is one copy.

QPRINT (continued)

/COUNT — (CLI32 only.) Counts the number of files this command processes.

/DELETE — Deletes files after printing them.

/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.

/DOCUMENTNAME=string — Prints string in block letters near the center of the header and trailer pages if any exist. If you omit this switch, the system prints the filename. Both /DESTINATION= and /DOCUMENTNAME= strings are always printed.

/END=n — Stops printing the file at page n. If you also use /BEGIN=, the ending page number must not be less than the starting page number. If you omit this switch or do not specify a page number, printing ends at the last page.

/FOLDLONGLINES — Continues printing long lines on the next line of the listing; by default, the system truncates them at the number of characters per line set for the printer.

/FORMS=forms–filename — Prints pathnames as specified by the forms control file, forms–filename. The file must be in directory :UTIL:FORMS. Check with your operator for the special forms that are available. If you omit this switch, standard forms are used. (You can create forms control files with the FCU utility.) Your job will wait in the queue until the system operator tells the printer to accept the form by giving EXEC's FORMS or DEFAULTFORMS command. Also, see the /OPERATOR switch.

/HOLD — Holds this job in the queue. You can later use the QUNHOLD command to release the job.

/MAPPER=mapper–filename

Names a mapper file that contains specifications to declare the ASCII values in pathname (one of the arguments to the QPRINT command) that the system should replace by one or more ASCII values. The replacement occurs within the system; pathname is not changed. Some mapper files (such as UPPER for lower–to–uppercase conversion) come with the operating system.

For example, a mapper file can contain a specification to have the system replace all occurrences of the ASCII value 176 (the tilde, ~) by the word "tilde"; as a result, whenever "~" is in pathname, "tilde" is printed. Mapper files also can specify overstriking so that, for example, "/" on top of c produces the cent symbol. Another mapper file application is to print non–English characters such as the German umlaut. A mapper file could have the code for an umlaut print correctly so that, for example, the word *loeschen* (delete) would appear in its original seven German characters rather than with *oe* representing an *o* with an umlaut.

/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. Use this switch if the job requires special forms or operator assistance.

/PAGES*[=n]* — (Print jobs) Does not print more than n pages. This switch applies only if the operator has specified a page limit for the queue. If the operator has specified a page limit and you use /PAGES without the value n, the system estimates the number of pages in your job as follows:

pages = (bytes–in–file)/1000 + 4

If the operator has set a page limit and the value you specify exceeds that limit, the job will not be processed.

/PASSTHRU — Prints in binary mode. Tells the printer cooperative process to send all characters, including those with value 377 octal, to the printer. For this switch to work, the operator must have enabled binary mode on the target printer (the printer served by the queue you specify). The operator can enable binary mode with the EXEC command BINARY. See also the /BINARY switch, which tells the printer to interpret characters of 377 octal.

/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.

QSNA *Command*

Submits a job to the Systems Network Architecture (SNA) queue.

Format

QSNA pathname *[...]*

This command places an entry in the Systems Network Architecture (SNA) queue. This type of queue provides a batch–type data transfer over a Data General SNA network to an IBM Remote Job Entry (RJE) system.

NOTE: The command does not actually send the specified file, but only provides the file pathname to the queue. Do not modify or delete the file until you are sure it has been sent.

If you do not use the /QOUTPUT= switch, the system creates a generic output file for the QSNA command in the form

:UDD:username:RJE.OUTPUT.n

where n is the job sequence number assigned to the job when you execute the command.

For more information about SNA/RJE, see the *SNA/RJE Operator's and User's Guide*, 093–000301.

- Accepts templates.
- No argument switches.
- Requirement: *Standard*.
- See also: QCANCEL, QDISPLAY, QHOLD, QMODIFY.

Why Use It?

Use the QSNA command to send a job through an SNA network.

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=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. Use the format +hh:mm:ss to specify a delay from the current time.

QSNA (continued)

/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–mmm–yy:hh:mm:ss), date (dd–mmm–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–mmm–yy:hh:mm:ss), date (dd–mmm–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.

/CONCATENATE
Sends the data from the specified sources as a single continuous data stream.

/COUNT
(CLI32 only.) Counts the number of files this command processes.

/DELETE
Deletes the source file(s) after completing the transfer. Use this switch only if the filename is not a device name.

/ERCL=n
Sets the exchange record length to n bytes where n is an integer from 0 through 255.

/HOLD
Holds this job in the queue. You can later use the QUNHOLD command to release the job.

/MEDIUM=devclass
Specifies the device class of the device or file whose data you are submitting to the SNA queue. The devclass choices are CONSOLE and EXCHANGE.

/NORESTART
If the system fails while processing this job, this switch does not restart it (at the beginning of the file) 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.

/QOUTPUT=pathname
Sets the generic output file of the batch process to this pathname (which may not be a queue name). By default, output goes to the BATCH_OUTPUT queue.

QSNA (continued)

/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 in your user profile. If you omit this switch, the system assigns a priority based on this formula: n = (m + 255) / 2,, where m represents the maximum queue priority specified in your user profile.

/QUEUE=queuename
Submits the job to the specified queue instead of to the default queue. The queue type must be SNA.

/SORT
(CLI32 only.) Sorts alphabetically the filenames this command processes.

/STREAM=n
Specifies the SNA/RJE stream number that will process the request, where n is in the range of 0 through 6. The default stream number is 1. If you specify 0, SNA/RJE can use any stream.

/SUBADDRESS=n
Sets to n the subaddress of the device or file whose data you are submitting to the SNA queue. The number can be from 0 through 15; the default is 0.

/TRANSPARENT
Sends the data to the IBM host as transparent data; that is, the host does not interpret the data.

/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.

/TYPE=sna-type
Directs SNA/RJE to read data from the device or file as specified by sna type. You can specify these types:

LINE	Performs data–sensitive read (default); translates ASCII data to EBCDIC. The default record length is 80 bytes.
SEQUENTIAL	Performs a dynamic read of the number of characters specified when SNA/RJE was configured; translates ASCII data to EBCDIC. The default record length is 80 bytes.
EBCDIC	Performs dynamic read of the number of characters specified when SNA/RJE was configured; does not translate data. The default record length is 80 bytes.

QSNA (continued)

/V	Displays the pathname of each queued file. This switch is helpful if you used a template argument.
/VERIFY	(CLI32 only.) Same as /V.

QSNA Example

```
) QSNA/MEDIUM=EXCHANGE/ERCL=100/TYPE=SEQUENTIAL JOB1 ↵
```

This command queues file JOB1 on the SNA queue. It transfers data in JOB1 to a remote IBM host. JOB1 contains a job for the host to execute. The command also sets the medium to exchange, and reads the file with an exchange record length of 100 characters using dynamic reads.

QSUBMIT *Command*

Submits a job to a batch or spool queue.

Format

QSUBMIT pathname *[...]*

This command takes the CLI command(s) in each file specified by pathname and submits them as a batch job for batch processing. Each file must contain the CLI commands you want to run as a batch job — including, if needed, working directory, searchlist, and default ACL.

(*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.)

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 templates.
- No argument switches.
- Requirement: *Standard*.
- See also: !LOGON, QBATCH, QDISPLAY, QCANCEL, QHOLD, QMODIFY, QUNHOLD.

Why Use It?

Use the QSUBMIT command to submit a job that requires more than one CLI command to either a batch or spool (print) queue. If the job you want to run requires just one CLI command, you may want to use the QSUBMIT command instead.

Do not use QSUBMIT to submit batch jobs in stacked format, which commonly applies to jobs submitted on punched cards on AOS/VS systems. See Appendix C.

Using a Remote Queue

If you want to submit a job to a remote queue (to use a remote printer, for example), use the /QUEUE= switch and specify the remote queue in the format

/QUEUE=hostname:queuename

You must use a full network pathname for a file that you submit to a remote queue unless the file resides on the same host as the queue.

QSUBMIT (continued)

If you use /QLIST= or /QOUTPUT= with a remote queue, always specify a full network pathname. The /NOTIFY switch will not work with a remote queue.

You can use a remote queue only if you have a profile on the remote system with the same username/password combination as your profile on the local system. Also, both systems must have the same password for the operator (OP) process.

NOTE: You should not submit large jobs to a remote queue; a failure in the network during processing of the job can cause a problem with the remote site's XLPT program, requiring its queue to be purged.

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=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, to specify after 2:00 p.m. on February 12, 1991, use /AFTER=12–FEB–91:14.

  Or, you can use the form +hh:mm:ss alone to specify a delay from the current time. For example, /AFTER=+3 executes the command as soon as possible, 3 hours from now.

/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–mmm–yy:hh:mm:ss), date (dd–mmm–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–mmm–yy:hh:mm:ss), date (dd–mmm–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.

/BINARY
: Prints in binary mode. This switch is valid only for devices that have binary mode enabled. Check with your operator for available local binary devices.

QSUBMIT (continued)

/COUNT	(CLI32 only.) Counts the number of files submitted.
/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. If you omit this switch, the system limits CPU time to 1 minute. The /CPU 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 time over that limit, the system will not process your job; it will wait in the queue.
/DELETE	Deletes the file(s) after processing them.
/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.
/JOBNAME=name	(Batch jobs only) 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.
/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.
/OPERATOR	Runs this job only if a system operator is on duty. Use this switch for a batch job that contains a MOUNT request.
/QLIST=pathname	Sets the generic list file of the batch process to this pathname (which may not be a queue name). By default, the list file has the form username.LIST.sequence_number.
/QOUTPUT=pathname	Sets the generic output file of the batch process to this pathname (which may not be a queue name). By default, output goes to the BATCH_OUTPUT queue.

QSUBMIT (continued)

/QPRIORITY=n — Assigns priority n to this job. The highest priority is 1; the lowest is 255. You cannot assign a priority 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 this job to the specified queue. If you omit this switch, BATCH_INPUT is used.

/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.

/SORT — (CLI32 only.) Sorts alphabetically the filenames this command processes. To see the names, you must also use /V.

/STRING — (CLI32 only.) Same as /S.

/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

QSUBMIT/TYPE=\CPD #: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.

QSUBMIT/TRAVERSE=\CPD #:PROJ.–

/TYPE=typecode — (CLI32 only.) Specifies one or more types of files to process. Valid values of type codes are in Table 2–8.

/V — Displays the names of the queued files. This switch is helpful if you used a template argument.

/VERIFY — (CLI32 only.) Same as /V.

/XW0=n — Places the value n (a decimal number) in the ?XXW0 word of the ?EXEC system call packet.

/XW1=n — Places the value n (a decimal number) in the ?XXW1 word of the ?EXEC system call packet.

/XW2=n — Places the value n (a decimal number) in the ?XXW2 word of the ?EXEC system call packet.

/XW3=n — Places the value n (a decimal number) in the ?XXW3 word of the ?EXEC system call packet.

QSUBMIT Example 1

```
) CREATE/I MYJOB ↵
)) DIRECTORY :UDD:TERRY:PROJECTS ↵
)) SEARCHLIST :UTIL,:F77 ↵
)) F77/DEBUG=STEP ACHIEVE ↵
)) F77/DEBUG=STEP CORRECT ↵
)) F77/DEBUG=STEP GRADE ↵
)) F77/DEBUG=NOSTEP REPORTS ↵
)) F77LINK/DEBUG ACHIEVE CORRECT GRADE REPORTS ↵
))) ↵
) QSUBMIT MYJOB ↵
Queued, Sequence number = 68, Qpriority = 127
```

The CREATE/I command creates a file and inserts text in it. Here the commands compile and link FORTRAN 77 source files. The single right parenthesis ends input. The QSUBMIT command submits all the commands to the default batch input queue, BATCH_INPUT.

This command submits FILE1 and FILE2 to the BATCH_INPUT queue.

QSUBMIT Example 2

```
) QSUBMIT/NORESTART/HOLD FILE1 FILE2↵
Queued, Sequence number = 169, Qpriority = 127
Queued, Sequence number = 170, Qpriority = 127
```

This command submits FILE1 and FILE2 to the BATCH_INPUT queue, and holds it until a subsequent QUNHOLD command releases it. If the system fails while processing this job, the job will not be restarted when the system comes up.

QSUBMIT Example 3

```
) QSUBMIT/AFTER=12:30 BATCHJOB ↵
QUEUED, SEQ=667, QPRI=127
```

The system will not process this job before 12:30 p.m.

QSUBMIT Example 4

```
) QSUBMIT/QUEUE=TYCHO:BATCH_INPUT/QLIST=:NET:OZ:UDD:MARCO:LIST_FILE&
↵
&) /QOUTPUT=:NET:OZ:UDD:MARCO:OUTPUT_FILE :NET:OZ:UDD:COMMON:XJOB ↵
```

This command submits a batch job to a queue on a remote host called TYCHO. The arguments with the /QLIST= and /QOUTPUT= switches use full network pathnames so the remote host can find the local files. A full network pathname is used for the job file because it does not reside on the same system as the remote queue.

QUNHOLD *Command*

Frees an entry currently held in a queue.

Format

QUNHOLD { sequence–number / jobname / , } *[...]*

This command releases for processing a job that you held in its queue by issuing a QHOLD command. You can release all jobs held under your username by using a comma to represent a null argument.

NOTE: You can only use this command for your own queue requests. It does not override an entry that was held by a system operator command to EXEC.

- No templates.
- No argument switches.
- Requirement: *Standard.*
- See also: QCANCEL, QDISPLAY, QHOLD, QMODIFY.

Why Use It?

Use the QUNHOLD command to process a job that has been held in its queue.

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=.

QUNHOLD Example

```
) QBATCH/JOBNAME=MYJOB  XEQ  MYPROGRAM ↵
Queued, Sequence number = 25, Qpriority = 127
...
) QHOLD  MYJOB ↵
...
) QUNHOLD  MYJOB ↵
```

The QHOLD command holds MYJOB in its queue. The QUNHOLD command releases MYJOB for processing. The argument could have been the job sequence number, 25, instead of the assigned job name, MYJOB.

RDOS *Utility*

Reads or writes an RDOS dump file or disk.

Format

XEQ RDOS command *[argument ...]*

RDOS (Real–time Disk Operating System) is another Data General operating system. This utility helps you exchange files between RDOS systems and AOS/VS or AOS/VS II systems. You specify one of the following commands to perform the action described.

DUMP	Dumps files in your working directory to a tape in RDOS format.
GET	Gets files from an RDOS disk and places them in a subdirectory.
LIST	Lists the files on an RDOS disk.
LOA	Loads an RDOS dump file from tape into your working directory.
PUT	Places files on an RDOS disk.

Do not try to initialize the RDOS disk before running this program.

- Accepts templates for pathname and filename arguments.
- Accepts utility switches (described later).
- Requirement: *Standard* (Execute access to the program file in :UTIL).
- See also: CONVERT.

Restrictions

The RDOS utility will *not* do the following:

- Dump to, or load from paper tape, cassette, or 7–track magnetic tape.
- Access disks that AOS/VS or AOS/VS II does not support (for example, Models 4047A, 4047B, 4237, 4238, 4048A, and 4057A).
- Load files onto the RDOS disk; the LOAD command loads tape files to an AOS/VS or AOS/VS II disk only.
- Dump files from the RDOS disk; the DUMP command dumps files from the AOS/VS or AOS/VS II disk only.
- Copy files as the RDOS XFER or AOS/VS and AOS/VS II COPY commands do.

You can use this utility only if the VSGEN procedure defined the disk unit that holds the RDOS disk as part of your system.

You can use AOS/VS and AOS/VS II templates, but not RDOS templates.

Why Use It?

Use this utility to transfer files between an RDOS system and an AOS/VS or AOS/VS II system. You can load files from an RDOS tape, or dump files to tape in RDOS format. You can also move files to and from an RDOS disk.

DUMP Command

This command dumps one or more AOS/VS–AOS/VS II files to an RDOS dump file. If you specify a link file, its resolution file is dumped (if it exists). You can use templates with filenames. You cannot dump peripherals, IPC files, queue files, or generic files.

NOTE: The RDOS utility does not add the S attribute to the resulting RDOS file if the AOS/VS or AOS/VS II filename ends in .PR or .SV. You will have to use the CHARACTERISTICS CLI command on your RDOS system to give the file an S attribute.

The utility will not make overlay files contiguous. You will have to use the local /C switch on the XFER command after you have loaded the dump file.

Format

XEQ RDOS DUMP rdos–dumpfile *[aos/vs–pathname] [...]*

DUMP Switches

/A	Aborts on an ABORT condition.
/L	Writes filenames to the current list file.
/L=pathname	Writes filenames to the specified file.
/V	Verifies each file that is transferred.

Argument Switches

/C	Converts NEW LINE characters to carriage returns.
/N	Does not transfer files matching this template.

RDOS DUMP Example

```
) XEQ RDOS DUMP/V @MTB0:1 +/C ↵
```

This command dumps all files in the working directory to file 1 of @MTB0. It converts all NEW LINEs to Carriage Returns, and lists the names of the dumped files on @OUTPUT. The dump file will be in RDOS format. All NEW LINEs will be converted, even those in .OB and .PR files.

GET Command

This command gets one or more files from an RDOS disk and places them in a subdirectory of your working directory. You can use templates with the RDOS filenames. The subdirectory must exist when you issue this command. Do not try to initialize the RDOS disk.

GET Command Format

XEQ RDOS GET/DISK=rdos-diskunit *[rdos-filename] [...]*

GET Switches

/A	Aborts on an ABORT condition.
/L	Writes filenames to the current list file.
/L=pathname	Writes filenames to the specified file.
/N	Does not load, just verifies the filenames.
/T	Moves the contents of all RDOS directories designated in the command line. Without this switch, the directories will be created, but no files will be placed in them.
/V	Verifies each file that is transferred.

Argument Switches

/C	Converts Carriage Returns to NEW LINEs.
/N	Does not transfer files matching this template.

NOTE: @ becomes the full pathname of the working directory. Explicit directory names, such as DP0 are not changed into their corresponding AOS/VS or AOS/VS II logical disk unit (LDU) names.

RDOS GET Example

```
) CREATE/DIR  SOURCES ↵
) XEQ RDOS  GET/V/DISK=@DPF2   SOURCES:+.SR/C ↵
```

This command copies all files that have .SR suffixes from RDOS subdirectory SOURCES of disk @DPF2. The RDOS utility changes all Carriage Returns in the .SR files to NEW LINEs. The directory SOURCES must exist in the working directory for this command to work.

LIST Command

This command lists the names of all files on an RDOS disk. You can use templates with the RDOS filenames. Do not try to initialize the RDOS disk. The command is equivalent to the RDOS CLI command LIST/E/A.

Format

XEQ RDOS LIST/DISK=rdos–diskunit *[rdos–filename] [...]*

LIST Switches

/A	Aborts on an ABORT condition.
/L	Writes filenames to the current list file.
/L=pathname	Writes filenames to the specified file.
/T	Lists the contents of all directories designated in the command line. Without this switch, the command lists the specified directories, but not the files that they contain.
/V	Verifies each file.

Argument Switches

/N	Does not list files matching this template.

NOTE: The LIST command will not list the SYS.DR of any directory unless it is explicitly listed as an RDOS filename in the command line. In addition, if there is a template in the command line, SYS.DR will not be listed regardless of whether or not you explicitly designate it.

RDOS LIST Example 1

```
) XEQ RDOS LIST/DISK=@DPF3  Z:–.– ↵
```

This command lists all files matching the template –.– from RDOS subdirectory Z on RDOS disk @DPF3. Notice that the template –.– matches only files with suffixes. The AOS/VS and AOS/VS II template –.– corresponds to the RDOS template –.*–.

RDOS LIST Example 2

```
) XEQ RDOS LIST/DISK=@DPD5  Z:+↵
```

This command lists all filenames in RDOS subdirectory Z on RDOS disk @DPD5. The AOS/VS and AOS/VS II template + corresponds to the RDOS template –.–.

RDOS LIST Example 3

```
) XEQ RDOS LIST/DISK=@DPF3/T ↵
```

This command lists the filenames of all files on RDOS disk @DPF3, including the filenames in each subdirectory.

LOAD Command

This command loads one or more files from an RDOS dump file. The directory filename suffix (.DR) is dropped from directory files but not from MAP.DR or user files that happen to use the .DR suffix.

Format

XEQ RDOS LOAD rdos–dumpfile *[...]*

LOAD Switches

/A	Aborts on an ABORT condition.
/	Loads the new file after deleting any file with the same filename.
/L	Writes filenames to the current list file.
/L=pathname	Writes filenames to the specified file.
/N	Does not load, just verifies the filenames.
/V	Verifies each file that is transferred.

Argument Switches

/C	Converts Carriage Return characters to NEW LINE characters. This is mandatory for text files; if you do not do it, you will not be able to read the file under AOS/VS or AOS/VS II.
/N	Does not transfer files matching this template.

RDOS LOAD Example 1

```
) XEQ RDOS LOAD @MTB0:0 +.SR/C +.RB ↵
```

This command loads all the files, whose names end in .SR or .RB, from file 0 on @MTB0 into the working directory. It converts all Carriage Returns in the source files (.SRs) to NEW LINEs; it does not convert Carriage Returns in the relocatable binary files (.RBs). Whoever gives this command probably will consider running the CONVERT utility for each of the newly loaded .RB files. This utility converts them to AOS/VS and AOS/VS II object (.OB) files.

RDOS LOAD Example 2

```
) XEQ RDOS LOAD/V @MTJ0:1 +.RB/N +.SV/N ↵
```

This command loads all files on @MTJ0:1 except those whose names end in .RB, .SV.

PUT Command

This command puts one or more files on an RDOS disk. You can put files into one subdirectory or partition only. This command uses the primary partition of the RDOS disk if you omit /DIR. It will not create subdirectories, subpartitions, or links on the RDOS disk. Nor will the command delete files on the RDOS disk. Do not try to initialize the RDOS disk.

If the AOS/VS or AOS/VS II file has a maximum index level of 0, then the resulting RDOS file will be a contiguous file. If the AOS/VS or AOS/VS II file's maximum number of index levels is not zero, and the file will fit into one block on the RDOS disk, then the resulting RDOS file will be a sequential file. If neither of the above conditions holds, then PUT will create a random RDOS file.

Format

XEQ RDOS PUT/DISK=rdos–diskunit [/*DIR=rdos–subdirectory] [aos/vs–filename] [...]*

PUT Switches

/A	Aborts on an ABORT condition.
/DIR=pathname	Puts files into the specified RDOS subdirectory.
/L	Writes filenames to the current list file.
/L=pathname	Writes filenames to the specified file.
/V	Verifies each file that is transferred.

Argument Switches

/C	Converts NEW LINE characters to carriage returns.
/N	Does not transfer files matching this template.

RDOS PUT Example

```
) XEQ RDOS PUT/V/DISK=@DPD5/DIR=SUBDIR  MYPROG.FR/C ↵
```

This command copies file MYPROG.FR from the AOS/VS or AOS/VS II working directory to RDOS subdirectory SUBDIR on RDOS disk @DPD5. It also converts NEW LINEs to Carriage Returns (CRs).

READ *Command*

Reads and displays a data–sensitive record from a file (CLI32 only).

Format

READ/FILEID=file–ID

This command gets and displays the next sequential record (the next line of text) from a file with text lines that are terminated by data–sensitive delimiters. The file is one you previously opened with the OPEN command. This OPEN command displayed the file–ID. You must give the file–ID to the READ command via the /FILEID= switch.

If you want to read all the records in a file you must be able to detect the end of file. The /EOF switch does this by returning a string of your choice, such as ***, when a READ command tries to read past the last record. Without this switch the READ command returns the null string when it tries to read past the last record.

To discover the file–IDs of all files you have opened, enter the OPEN command without an argument.

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

Why Use It?

Use the READ command to read and display the next sequential line in a data–sensitive file you have previously opened. This command eliminates the need to construct files whose records end with the two characters & (ampersand) and NEW LINE so that the CLI can read past the first line in these files.

READ 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=.

/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 command /EOF=string switch.

If you don't specify this switch, the system uses the one you specified for the OPEN commands /EOF=string switch.

If you don't specify this switch here or for OPEN, the system returns the null string when the READ command encounters the end of the file.

/FILEID=file-I

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 Examples

```
) OPEN/READ/EOF=DONE  MODULE_LIST ↵
MODULE_LIST

) READ/FILEID=MODULE_LIST ↵
main.c

) READ/FILEID=MODULE_LIST↵
last.c

) READ/FILEID=MODULE_LIST ↵
DONE

) READ/EOF=***/FILEID=MODULE_LIST↵
***
```

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

!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.
```

RELEASE *Command*

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

RENAME pathname { new–filename / full–pathname[1] }

[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.

086–000200 updates
093–000646

A Word of Caution

Many Data General software products expect certain files and directories to have a specific name. For this reason you should not rename any directory or file supplied by Data General unless the related documentation or Release Notice specifically advises you to do so.

RENAME 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=.

RENAME Example 1

```
) RENAME DATB.90  DATA.90 ↵     (Change a filename  in the working
                                          directory that was entered incorrectly)

) RENAME  :UDD:ROBIN:DATA.90  DATA.91 ↵ (Change a filename that is not in the
                                          working directory.)
```

RENAME Example 2

```
) RENAME  MYFILE  :UDD:CHRIS:MYDIR:MYFILE ↵
```

The command above renames the pathname of — that is, transfers — file MYFILE to directory :UDD:CHRIS:MYDIR. MYFILE no longer exists in the working directory.

RENAME Example 3

```
)  RENAME  :UDD:CHRIS:WORK:SOURCES  :UDD:CHRIS:SOURCES ↵
```

The RENAME command above transfers directory UDD:CHRIS:WORK:SOURCES and all its files to directory :UDD:CHRIS:SOURCES.

RENAME Example 4

```
)  FILES/S  +.LS ↵                    (Lists names starting with .LS.)

Directory :UDD:ALLAN:XDIR
ABC.LS   MOD1.LS   ZORK.LS

)  RENAME ([!FILENAMES/NOEQUAL +.LS]) &↵          (Transfers the files
&  :UDD:AL:LIST_DIR:([!FILENAMES/NOEQUAL +.LS]) ↵  to directory LIST_DIR.)
```

The command above renames multiple files to another directory.

RENAME Sample Macro

The next example shows a macro called XFER.CLI. The macro lets you use templates and most of the MOVE command switches with RENAME. The macro assumes you will use it from the parent directory of the file(s) you want to transfer.

```
[!equal,(%2%),()]
  write XFER macro - transfers files to another directory.
  write Usage: XFER  directory-pathname   filename-or-template ...
  write Switches:
  write /AFTER/TLA=
  write /AFTER/TLM=
  write /AFTER/TCR=
  write /BEFORE/TLA=
  write /BEFORE/TLM=
  write /BEFORE/TCR=
  write /COUNT
  write /L{=pathname}
  write /Q
  write /TRAVERSE=
  write /TYPE=
  write /SORT=
  write /V
  write /1 /2
  [!else]
     [!equal,%0/DOIT%,]
     push; prompt pop
     var0 0
    searchlist [!edirectory [!path %0\%.cli]]
     string [!file/noequal%0/after/before/tla/tcr/tlm/traverse/type/sort% %2-%]
     [!equal,([!string]),()]
      write ERROR: No files match template: %2%
     [!else]
     [!equal,([!path %1%]),()]
          write ERROR: Directory does not exist: %1%
       [!else]
          XFER/DOIT%0/% %1% ([!string])
       [!end]
     [!end]
     [!equal,%0/count%,/count]write Transferred [!var0]
files.[!end]
     pop
     [!end]
     [!ne,%0/DOIT%,]
     rename%0/0/1/q%  %2%  %1%:%2%
     var0 [!uadd [!var0],1]
      [!equal,(%0/v%),(/v)]write%0/1%  %2% transferred to %1%:%2%.[!end]
     [!end]
     [!end]
```

RENAME Sample Macro (continued)

The format for running the macro is

XFER full-destination-directory-pathname filename-or-template *[...]*

An example of the macro follows:

```
) XFER/V/COUNT/SORT :UDD:CHRIS:WORK:SOURCES +.C +.F77 <NL>

HLITS.C transferred to :UDD:CHRIS:WORK:SOURCES:HLITS.C
LITERALS.F77 transferred to :UDD:CHRIS:WORK:SOURCES:LITERALS.F77
SUB.C transferred to :UDD:CHRIS:WORK:SOURCES:SUB.C
SUB.F77 transferred to :UDD:CHRIS:WORK:SOURCES:SUB.F77
Transferred 4 files.
```

As with the other cases where RENAME specifies a different directory, the files renamed no longer exist in the working directory.

REVISION *Command*

Displays or sets a program revision number.

Format

REVISION pathname *[field1[.field2[.field3[.field4]]]]*

This command either displays the current revision number assigned to the specified program, or lets you assign a revision number to the program. You can omit the .PR suffix from the program pathname.

The revision number consists of as many as four 3–digit fields, each of which can range from 0 through 255. (If you specify a single digit for a field, a zero will appear before that digit.) By convention, Data General uses *field*1 and *field*2 for system program revision numbers; for example, 7.69.

To display or set revision numbers, you can use templates in the pathname. Generally, compilers and assemblers assign their own revision numbers to object files, and this revision number moves into the program file.

- Accepts templates for pathname.
- No argument switches.
- Requirement: *Standard*.

Why Use It?

Use the REVISION command to find the revision number of a program. You can use this command to check the revision of a software package, perhaps to ensure that your system is running the appropriate software.

If you are developing software, you can assign a revision number to each version of a program. This helps you keep track of program development.

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–mmm–yy:hh:mm:ss), date (dd–mmm–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.

REVISION (continued)

/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–mmm–yy:hh:mm:ss), date (dd–mmm–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 this command processes.

/SORT — (CLI32 only.) Sorts alphabetically the filenames this command processes.

/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

```
REVISION/TYPE=\CPD #:PROJ–.PR 3.04
```

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

```
REVISION/TRAVERSE=\CPD #:PROJ–.PR 3.04
```

/TYPE=typecode — (CLI32 only.) Specifies one or more types of files to process. Valid values of type codes are in Table 2–8.

/UNX — (CLI32 only.) Sets the revision level of a UNIX program file (file type UNX). The CLI will let you display, but not set, the revision number for a UNX file without this switch.

CAUTION: The CLI will write the revision numbers you specify into locations 410 and 411 (octal) of the file, whether or not it is a program file. Be sure that the file is a program file, since program files reserve locations 410 and 411 for revision numbers but other file types use them for normal information storage.

/V — Displays the program name.

/VERIFY — (CLI32 only.) Same as /V.

REVISION Example 1

```
) REVISION  EXEC.PR ↵
02.10.00.00
```

This command displays the revision number of the EXEC program.

REVISION Example 2

```
) REVISION  MYPROG  2.10 ↵
) REVISION/V MYPROG ↵
MYPROG.PR  02.10.00.00
```

The first command assigns revision number 2.10 to the program MYPROG. The second command displays the program name and its revision number.

(Note that to assign the revision number 2.1 to the program, the argument must appear as either 2.10 or 02.10.)

REVISION Example 3

```
) REVISION/V  +.PR ↵
MYPROG.PR   02.01.00.00
PROG2.PR    01.05.25.50
PROG1.PR    01.255.00.00
```

This command requests a display of the revision number for each program file in the working directory.

REVISION Example 4

```
) REVISION/V/AFTER/TLM=1-JAN-90   +.PR  2.00.00.00 ↵
MYPROG.PR
SORT_APPL.PR
UPDATE_NAMES.PR
```

This command, for CLI32 only, sets the revision number of all program files modified after New Year's Day, 1990 to 2.00.00.00.

REWIND *Command*

Rewinds one or more magnetic tapes.

Format

REWIND { tape–unit–name / linkname } *[...]*

This command rewinds the specified tape(s). You can specify the tape either by the name of the tape unit it is mounted on, or by the linkname you used in a MOUNT command.

- Accepts templates.
- No argument switches.
- Requirement: *Standard,* but to issue the command with the /TRESPASS switch you must have System Manager privilege.

Why Use It?

Use the REWIND command to return to the beginning of a magnetic tape so that you can either dismount the tape or write to it again. REWIND is handy after any command that reads or writes a tape : COPY, CPIO_VS, DUMP_II or DUMP, LOAD_II or LOAD.

You can also issue this command simply to check whether or not a tape unit is on line.

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=.

/TRESPASS — Rewinds a tape unit that you share with another system through a MRC (Message–based Reliable Channel) I/O subsystem. This switch overrides another system's use of the tape unit, so you must have System Manager privilege.

Use this switch if another system on an MRC I/O subsystem has opened the tape unit and then failed. This switch both rewinds the tape unit and removes the other system's exclusive access to it. Now, you can use the tape unit as you want.

REWIND Example 1

```
) LOAD_II/V @MTD1:0 ↵

...(verifies files loaded)...

) LOAD_II/V @MTD1:1 ↵

...(verifies files loaded)...

) REWIND @MTD1 ↵
```

The first command loads into the working directory the contents of the first file on the tape mounted on unit MTD1. The second command loads the contents of the second file on the same tape. The REWIND command returns to the beginning of the tape so it can be taken off the unit.

REWIND Example 2

The following two commands on an AOS/VS II system attempt, first unsuccessfully and then successfully, to use a tape unit on an MRC I/O subsystem. The user has System Manager privilege.

```
) LOAD_II/V @MRCTAPE000A01:0 ↵
Error: Device in use by another port

) REWIND @MRCTAPE000A01 ↵
Error: Device in use by another port

) REWIND/TRESPASS @MRCTAPE000A01 ↵
) LOAD_II/V @MRCTAPE000A01:0 ↵
```

The tape I/O proceeds.

REWIND Example 3

The following command lets you discover whether tape unit @MTD0 is on line.

```
) REWIND @MTD0 ↵
Warning: Physical unit off-line, File @MTD0
```

RUNTIME *Command*

Displays runtime information for a process.

Format

RUNTIME [*process–ID* | *processname* | *username:processname*] *[...]*

This command displays information about the specified process, including

- the amount of real time that elapsed since the process started
- the number of CPU seconds used by the process
- the number of I/O blocks (blocks written or read by the process)
- the number of 2–Kbyte memory pages multiplied by the CPU time (in seconds)

You can supply a process ID or a process name. If you supply a simple process name, the CLI assumes your username. You can use either process name form, but only if the process was created with the PROCESS command and /NAME=name switch.

If you omit the argument, the command returns information about the current CLI process.

- No templates.
- No argument switches.
- Requirement: *Standard.*
- See also: WHO, PED (explained in *Managing AOS/VS and AOS/VS II*).

Why Use It?

Use the RUNTIME command to obtain runtime information about a process. You may, for example, want to know how long a program has been running, or how long you have been logged on. Issue it against PID 2 (the master CLI process) to see how long the system has been running.

You can also issue the RUNTIME command repeatedly for a specific process to determine if it is active (that is, using CPU time or I/O blocks), or pending.

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.

RUNTIME Example 1

```
) RUNTIME ↵
Elapsed 2:23:49, CPU 0:00:03.155, I/O Blocks 327, Page Secs 410
```

This command shows runtime statistics for the current CLI process.

RUNTIME Example 2

```
) RUNTIME 15 ↵
Elapsed 4:11:21, CPU 0:00:02.728, I/O Blocks 396, Page Secs 275
```

This command shows runtime statistics for PID 15.

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.

SCOM (continued)

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.
/V	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.

SCREENEDIT *Command*

Displays or sets the current Screenedit mode.

Format

SCREENEDIT [*ON* / *OFF*]

With no argument, this command reports whether Screenedit mode is On or Off. With an argument, this command lets you turn Screenedit mode on or off. When you log on at a terminal, Screenedit mode is set to On (the default value). In batch (noninteractive) mode, the default setting for Screenedit mode is Off.

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

Why Use It?

Use this command to turn on or off your ability to edit a command line by using control characters. Because the Screenedit setting can differ from one environment level to another, you can use this command to display the current setting as you move from one level to another.

Screenedit Cursor Control

Screenedit mode, when turned on, allows you to use control character sequences to edit a command line. This editing includes moving the cursor back and forth on the command line, opening the line for character insertion, closing the line after character insertion, and erasing part or all of the command line. The following text describes the control characters you can use in Screenedit mode.

CTRL–A	Displays the last command line entered and makes it available for editing — one of the most frequently used features of Screenedit mode. Also, moves the cursor to the end of the current line.
CTRL–B	Moves the cursor to the end of the previous word on the line.
CTRL–E	Enters insert mode. After typing the characters to be inserted, press CTRL–E again to exit insert mode.
CTRL–F	Moves the cursor to the beginning of the next word on the line.
CTRL–H	Moves the cursor to the beginning of the current line (equivalent to pressing the HOME key).
CTRL–I	Inserts a tab (equivalent to pressing the TAB key).
CTRL–K	Erases everything to the right of the cursor (equivalent to the ERASE EOL key).
CTRL–X or —>(rightarrow key)	Moves the cursor one position to the right.
CTRL–Y or <— (leftarrow key)	Moves the cursor one position to the left.

SCREENEDIT 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=.

/LEVEL=n — Sets the Screenedit mode to that in 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 Screenedit mode to that of the previous environment level. (Omit the argument.) With *=n* (CLI32 only), sets the Screenedit mode to the one used in the previous CLI environment level. The *n* specifies the number of levels above the current level (toward 0).

/PREVIOUS[=n] — (CLI32 only.) Same as /P.

SCREENEDIT Example

```
) SCREENEDIT ↵
ON

) MOVE/FTA :NET:ATHENA:UDD:LEE  TESTFILE.+ ↵
```

Press CTRL–A to redisplay the previous command.

```
) MOVE/FTA :NET:ATHENA:UDD:LEE  TESTFILE.+ ↵
```

Press CTRL–H to move the cursor to the beginning of the line.
Press CTRL–F twice to move the cursor to the beginning of TESTFILE.+
Type a new filename over the previous argument.

```
) MOVE/FTA  :NET:ATHENA:UDD:LEE  TESTDATA.+ ↵
```

SEARCHLIST *Command*

Displays or sets the search list.

Format

SEARCHLIST *[directory-pathname] [...]*

This command either displays the current search list, or lets you set it.

When you refer to a file without giving a complete pathname (one that starts at the root directory), the system begins looking for the file in your working directory. If the file does not exist there, the system continues to look for the file within each directory that is named on your search list.

The search list should include directories that contain files that you frequently use. Specify the directories in the order that you want them searched. The maximum number of directories allowed on a search list is eight.

The search list is part of the CLI environment; you can set a different one for each level.

If you rename a directory that is on the current search list, the system will update the search list accordingly to include the new directory name. Do not rename a directory that is on a previous environment's search list; if you do this, when you return to the previous environment the CLI will display a *File does not exist* error message and retain the search list used in the previous environment.

NOTE: You cannot delete a directory that is on any search list of any active CLI process. Nor will the system allow release (RELEASE command) of any LDU that contains a directory on the search list of any active CLI process. In either case, to the DELETE or RELEASE command will return the error message *Directory in use.*

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

Why Use It?

The search list is an important tool to let the system find files you use frequently (like a text editor in directory :UTIL).

Use the SEARCHLIST command to display or set your current search list. Many people set the search list routinely in the startup macro that is executed when they log on.

SEARCHLIST 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=.

/INSERT=n	(CLI32 only.) Inserts the directory you specify as directory n in the search list; for example, SEARCHLIST/INSERT=3 MYDIR inserts MYDIR as the third directory in the search list.
/K	Deletes (kills) the current search list, if any. Omit the pathname arguments.
/LEVEL=n	CLI32 only.) Sets the search list to the one used in the specified environment level. 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 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 search list to the setting used in the previous environment. (Omit the pathname arguments.) With =*n* (CLI32 only), sets the search list to the one used in the previous CLI environment level. The *n* specifies the number of levels above the current level (toward 0).
/PREVIOUS*[=n]*	(CLI32 only.) Same as /P.
/REMOVE=n	(CLI32 only.) Removes the directory you specify as directory n from the search list.

SEARCHLIST Example 1

```
) SEARCHLIST ↵
:UDD1:TOM,:MACROS,:UTIL
```

This command displays the current search list.

SEARCHLIST Example 2

```
) XEQ  MYPROGRAM ↵
Error:  File does not exist

) SEARCHLIST [!SEARCHLIST]  :UDD:COMMON ↵

) SEARCHLIST ↵
:UDD1:TOM,:MACROS,:UTIL,:UDD:COMMON

) XEQ MYPROGRAM ↵
...
...

Program processing completed.
```

The first command tries to execute a program, but the system cannot find it in either the working directory or a directory on the search list. With CLI32, you could have used the command SEARCHLIST/INSERT=3 UDD:COMMON instead of the one used.

The second command resets the search list by adding the directory :UDD:COMMON to the current search list (represented by the !SEARCHLIST pseudomacro). The directory :UDD:COMMON is where the program file resides. The second XEQ command works because the CLI can now locate the program file.

SEARCHLIST Example 3

) SEARCHLIST ↵ *UTIL :NET:UTIL*	(Displays the search list.)
) PUSH ↵	(Pushes (descends) a level.)
) SEARCHLIST :UDD:KERRY ↵	(Sets the search list.)
...	(Does things that need the new search list.)
) SEARCHLIST/P ↵	(Resets the search list to that of the previous environment.)
) SEARCHLIST ↵ *:UTIL :NET:UTIL*	(Displays the search list.)
) POP ↵	(Returns to the previous level's environment.)

!SEARCHLIST *Pseudomacro*

Expands to the current search list.

Format

[!SEARCHLIST]

This pseudomacro represents a list of the directories that make up the current environment's search list.

- No arguments.
- Accepts a macro name switch (described later).
- Requirement: *Standard.*
- See also: SEARCHLIST.

Why Use It?

Use the !SEARCHLIST pseudomacro to include the current search list within a CLI command argument.

Macro Name Switches

/LEVEL=n — (CLI32 only.) Expands to the search list used in 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*, expands to the search list of the previous CLI environment level. With =*n* (CLI32 only), expands to the search list used in the previous CLI environment level. The *n* specifies the number of levels above the current level (toward 0).

/PREVIOUS*[=n]* — (CLI32 only.) Same as /P.

!SEARCHLIST Example 1

```
) PUSH ↵
) SEARCHLIST [!SEARCHLIST]  :UTIL:SPECIAL_PROGRAMS ↵
```

The first command pushes to the next lower CLI environment. The next command sets the search list for the current environment, adding the directory :UTIL:SPECIAL_PROGRAMS to the current search list. The pseudomacro frees you from having to type all the directories that make up the current search list.

!SEARCHLIST Example 2

(within a macro)

```
push
xeq/s  myprog
.
.
write The program termination message is [!string]
write The search list is [!searchlist]
pop
```

These macro commands report information following the termination of a program. The first statement displays the contents of the CLI String to which you can direct a program's termination message. The second statement displays the current search list. This information could be helpful in determining why a program terminated unexpectedly.

!SEARCHLIST Example 3

In this example, the CLI begins by evaluating !SEARCHLIST as the current search list (in this case :UTIL,:UDD:LEE), and writes the result.

```
) WRITE The current searchlist is [!SEARCHLIST] ↵
The current searchlist is :UTIL,:UDD:LEE
```

We then change the environment, and set :UDD:SANDY as the search list for that environment. The CLI now evaluates [!SEARCHLIST] as :UDD:SANDY for the current environment, and with the /P switch, as :UTIL,:UDD:LEE for the previous environment.

```
) PUSH; SEARCHLIST :UDD:SANDY ↵

) WRITE Now the current searchlist is [!searchlist] ↵
Now the current searchlist is :UDD:SANDY

) WRITE [!SEARCHLIST/P] ↵
:UTIL,:UDD:LEE
```

SEND *Command*

Sends a message to a terminal.

Format

SEND [*process–ID* | *processname* | *username:processname* | *termintal–filename*] message

This command sends a message in the form of a text string to a terminal specified by a process ID, process name, username:process name pair, or terminal filename (as in @CON6). To list processes running on your system, use the WHOS macro. The WHOS macro lists PIDs, process names, and terminal filenames.

You must supply a process ID or a process name. You can use any process name form, but only if the process was created with the PROCESS command and /NAME=name switch.

The CLI translates commas and tabs into spaces and does not send control characters.

To send a message to all processes, use the BROADCAST macro.

- Accepts a template for the console name.
- No argument switches.
- Requirement: *Standard.*
- See also: WHO.

Why Use It?

Use the SEND command to display a message on or send ASCII characters to a terminal. For example, you might want to notify another user of a meeting. You can use the WHOS macro to discover the PIDs of users on the system.

(If, as the system operator, you plan system shutdown, use the BROADCAST macro — which uses the SEND command — to to warn users. Or if you plan to print a long document, you might want to use BROADCAST to let others know that the line printer will be busy for awhile.)

NOTE: The /NRM switch on the CHARACTERISTICS command, when it is turned on, disables the reception of messages except those sent by PID 2. The system returns a *Message receive disabled* error message if your message is rejected for this reason.

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!
```

!SIZE *Pseudomacro*

Expands to the byte length of one or more files.

Format

[!SIZE pathname *[...]*[1]]

[1] Multiple arguments only with CLI32.

This pseudomacro returns the length of the specified file(s) in bytes. If a file is a link, the CLI returns the length of the the resolution file. If the CLI cannot find a file, it returns a length of 0 (not an error message) for that file.

With CLI32, the pseudomacro can return the length of more than one file.

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

Why Use It?

Use the !SIZE pseudomacro to include a file's length in a CLI command argument.

!SIZE Example 1

(within a macro)

```
WRITE The new size of the TEMP file is [!SIZE TEMP] bytes.
```

This command reports the current length of the file TEMP. The message will look something like this:

The new size of the TEMP file is 1145 bytes.

!SIZE Example 2

(within a macro)

```
[!ugt, [!size %1%], 20000]
        write file %1% exceeds 20000 bytes.  It is [!SIZE %1%] bytes long.
[!else]
        write File %1% contains 20000 bytes or fewer.
[!end]
```

These macro statements check the size of a file. If the file's size is greater than 20,000 bytes, the macro displays a message like this:

File MYFILE exceeds 20000 bytes. It is 34520 bytes long.

If the length of the file is 20,000 bytes or fewer, the macro writes a message like this:

File MYFILE contains 20000 bytes or fewer.

!SONS *Pseudomacro*

Expands to a list of subordinate process IDs.

Format

[!SONS [*process-ID* | *processname* | *username:processname* | *hostname:username:processname*]]

This pseudomacro returns the process ID (PID) of each son of the specified process. If you do not specify a process, the pseudomacro returns the PIDs of the current CLI's sons.

!SONS returns a process ID (PID) for every subordinate process. It returns the PIDs of direct subordinates only, not grandsons.

If your system is part of a XODIAC/XTS network, you can include a hostname with !SONS to learn about PIDs running on that system. An example is

) WRITE [!SONS CENTRAL:3] ↵

- No templates.
- No argument switches.
- No macro name switches.
- Requirement: *Standard,* but to refer to a remote host, you must have the same username/password combination on the local and remote system.
- See also: TREE, !PIDS.

Why Use It?

Use the !SONS pseudomacro to obtain numbers of all process IDs subordinate to a process. You can use this information as an argument to another command (see Example 2).

!SONS Example 1

) WRITE [!SONS] ↵
25 26

) WRITE [!SONS 1] ↵
2 3 4 5

) WRITE [!SONS TITIAN:1] ↵
TITIAN:2 TITIAN:3 TITIAN:4 TITIAN:5

!SONS Example 2

(within a macro)

TERMINATE [!SONS]

This command terminates all subordinate processes.

!SONS Example 3

```
) WRITE [!SONS OP:EXEC] ↵          (Request PIDs of EXEC's sons.)
7 10 14 15 17 22
```

!SONS Example 4

```
) WHO [!SONS OP:EXEC] ↵            (Request process information
                                    about EXEC's sons.)

PID:   7     OP      LPB     :UTIL:XLPT.PR
PID:   10    LINDA   CON3    :CLI.PR
PID:   14    TSAO    CON40   :CLI.PR
...    ...   ...     ...     ...

) WHO [!SONS CENTRAL:3] ↵          (Request process information about
                                    EXEC's sons on another system.)
```

SPACE *Command*

Displays disk space usage information for a directory, or changes the maximum allotment for a control point directory.

Format

SPACE [*control-point-directory [blocks]* | *LDU-filename* | *standard-directory*[1] | *non-directory-pathname*[1]]

[1] CLI32 with AOS/VS II only

This command displays disk space information (the maximum number of available blocks, how many are used, and how many remain) for a control point directory or an LDU (logical disk unit). For a control point directory (created with the command CREATE/MAX=n), you can set a new maximum size if you have write access to the parent directory. The CLI32 SPACE command under AOS/VS II can also display usage information for a standard directory (one created with CREATE/DIR) or the number of disk blocks consumed by a nondirectory file.

A *control point directory* is a type of directory that is allocated a fixed number of blocks. (A block is 512 bytes in length.) An error occurs if you perform any operation that would cause the contents of such a directory to exceed this maximum size. You can create a control point directory with the command form CREATE/MAX=n.

NOTE: The space shown for a control point directory includes blocks that the system requires. This is why a CPD that contains no files may show a certain number of blocks as being in use.

- Accepts templates.
- No argument switches.
- Requirement: *Standard,* but you must have Write access to change the space in a control point directory.

Why Use It?

Use the SPACE command to discover how much disk space is available, how much of that is being used, and how much still remains. This information helps you monitor disk space and gives you an indication of when you should delete unnecessary files.

You can use this command to change the maximum size of a control point directory. For example, when moving or copying files to a control point directory, you may reach its capacity. You can then use this command (first form above) to increase the size of the directory so that it can accommodate the files. (Space actually used by all your files cannot, however, exceed the disk quota granted in your user profile.)

You can also use this command with a logical disk unit to determine how much space remains on the LDU, and how much space has been consumed by each user.

With CLI16 and/or AOS/VS, you cannot use SPACE command to get information on a standard directory; if you try, the CLI will return the message *Not a control point directory*. Therefore, if you are using CLI16 and/or AOS/VS and space accounting is important, a control point directory has a major advantage over a standard directory. With CLI32 and AOS/VS II, the SPACE command *does* return the amount of space used in a standard directory; thus with CLI32 you can use a standard directory even if space accounting is important.

If, after creating and using a directory, you decide that the other type would be more suitable, create a directory of the desired type, move all files from the original directory to it, and the delete the standard directory.

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–mmm–yy:hh:mm:ss), date (dd–mmm–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–mmm–yy:hh:mm:ss), date (dd–mmm–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.

SPACE (continued)

/COUNT — (CLI32 only.) Counts the number of files this command processes.

/SORT — (CLI32 only.) Sorts alphabetically the filenames this command processes. To see the names, you must also use /V.

/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. Without this switch, a command such as

```
SPACE/TYPE=\CPD #:PROJ.DIR
```

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

```
SPACE/TRAVERSE=\CPD #:PROJ.DIR
```

give expected results.

/TYPE=typecode — (CLI32 only.) Specifies one or more types of directory files to process. The pertinent types are DIR, LDU, and CPD.

/V — Display the name of the directory.

/VERIFY — (CLI32 only.) Same as /V.

SPACE Example 1

```
) SPACE ↵
MAX 10000, CUR 8716, REM 1284

) COPY DUMPFILE :UDD:TOM:HELP.DIR:HELPDUMP ↵
Error: Control Point Directory max size exceeded, File dumpfile
copy,dumpfile,:udd:tom:help.dir:helpdump

) SPACE = 20000 ↵
) COPY/D DUMPFILE :UDD:TOM:HELP.DIR:HELPDUMP ↵
```

The first command shows the maximum amount of space (10,000 blocks) available to the working directory (a control point directory), and the number of blocks currently used and the number remaining.

The next command tries to copy a file to this directory, but the file causes an error because its size would exceed the directory maximum.

The second SPACE command increases the number of blocks allocated for the directory. The COPY command (with the /D switch to delete any partial copy that was made) then succeeds.

SPACE Example 2

With CLI16 or AOS/VS, you cannot use SPACE command to get information on a standard directory. With CLI32 under AOS/VS II, the SPACE command *does* return the amount of space used in a standard directory. The following example compares the behavior of the two CLIs.

```
) WHO ↵

PID:  26 CHRIS  00026   :CLI16.PR          (Running CLI16)
) CREATE/DIR MYDIR ↵
) CREATE MYDIR:XFILE ↵
) SPACE MYDIR ↵

Warning: File is not a control point directory   (Error message)

) XEQ :CLI32 ↵                             (Run CLI32.)
AOS/VS II CLI...                           (CLI32 displays program banner.)

) SPACE MYDIR ↵
Max  NA, Cur  1, Rem  NA                   (CLI32 reports that 1 block is used.)
```

SPACE Example 3

With CLI16 or AOS/VS, you cannot use the SPACE command to get information on a non–directory file. With CLI32 under AOS/VS II, the SPACE command *does* return the amount of space used by any file for which you have access. The following example compares the behavior of the two CLIs.

```
) WHO ↵

PID:  26 PB  00026   :CLI32.PR             (Running CLI32)

) SPACE LOGON.CLI ↵
Max  NA, Cur  4, Rem  NA                   (CLI32 reports that 4 blocks are used.)

) XEQ :CLI16 ↵                             (Run CLI16.)
AOS/VS CLI...                              (CLI32 displays program banner.)

) SPACE LOGON.CLI ↵

Warning: File is not a control point directory   (Error message)
```

SQUEEZE *Command*

Displays or sets the Squeeze mode.

Format

SQUEEZE [*ON* / *OFF*]

This command either shows whether the Squeeze setting is currently on or off, or lets you set it to either value.

Squeeze mode, when turned on, causes the CLI to compress its output by combining multiple space or tab characters into a single space. (This does not, however, affect output from the TYPE command.)

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

Why Use It?

Use the SQUEEZE command if you want to turn Squeeze mode on or off. You might want to turn Squeeze mode on if you are using the /L switch to store CLI output in a file. Squeeze mode is useful for speeding up your terminal output or creating indirect files of filenames.

NOTE: Most CLI commands provide the /Q switch, which lets you turn on Squeeze mode for the duration of that command only. Thus, you can use Squeeze mode for individual commands without making a global change.

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=.

/LEVEL=n — CLI32 only.) Sets the squeeze mode to the one used in 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.

SQUEEZE (continued)

/P[=n]	Without =*n*, sets the squeeze mode to the one used in the previous environment. (Omit the pathname arguments.) With =*n* (CLI32 only), sets the squeeze mode to the one used at the specified environment level. The *n* specifies the number of levels above the current level (toward 0).
/PREVIOUS*[=n]*	(CLI32 only.) Same as /P.

SQUEEZE Example 1

```
) SQUEEZE ↵
OFF
```

This command shows the current Squeeze setting.

SQUEEZE Example 2

```
) SQUEEZE ON ↵
) FILESTATUS/NHEADER/TYPE=TXT/L=TEXTFILES ↵

) TYPE TEXTFILES ↵
=TX.CLI =JACKS =CAPTIONS =MORTGAGE_BATCH.CLI =TOOL_NOTES
=FRAG =STARTUP =FINALSPEC =MODEM_NUMBERS

) TYPE [TEXTFILES] ↵
...
(The contents of each file are displayed.)
...
```

The first command turns Squeeze on. The FILESTATUS command then builds a file called TEXTFILES, which contains the pathname of each text file in the working directory. Because Squeeze is on, the file pathnames are separated by a single space, as shown by the first TYPE command. The second TYPE command uses square brackets to display the contents of each file whose name appears in TEXTFILES.

STRING *Command*

Displays or sets the contents of the CLI String (CLI16 version — CLI32 version follows.)

Format

STRING *[argument] [...]*

This command either displays the current contents of the CLI String, or lets you replace the contents with a new text string. CLI16 has one unnamed string, whose maximum length is 127 characters.

- No templates.
- No argument switches.
- Requirement: *Standard.*
- See also: STRING (CLI32 version), !STRING.

Why Use It?

Use the STRING command to display (and set) the contents of the CLI String. When you execute a program via the XEQ or PROCESS command, you can use the /S switch to place the program termination message in the unnamed string, and then examine it with the STRING command.

STRING also lets you enter text and pseudomacro information into the CLI String.

The !STRING pseudomacro lets you use String contents as a command argument and display multiple–word strings without having commas inserted by the CLI.

The CLI32 STRING command, explained next, also offers an unlimited number of named strings; so if using strings in macros is important to you, use CLI32 and see the CLI32 description.

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.

/K	Clears the CLI String by setting its value to null. (Omit the argument.)
/P	Replaces the current String contents with the contents of the previous environment's CLI String. (Omit the argument.)

STRING (CLI16) Example 1

```
) COMMENT Clear the current string, whatever it is. ↵
) STRING/K ↵
) COMMENT Place a new value in the string. ↵
) STRING The working directory is [!DIRECTORY]. ↵
) COMMENT Display the current string. ↵
) STRING ↵
```
The working directory is :UDD1:CHRIS:SOURCES

STRING (CLI16) Example 2

```
) XEQ/S  MYPROG  SAMPLE ↵

) STRING ↵
```
Warning: File does not exist, File SAMPLE

The XEQ command included the /S switch to store the program termination message in the CLI String. The STRING command displays the contents of the CLI String after the program terminates.

STRING (CLI16) Example 3

(within a macro)

```
push
string [!read Do you want to print file %1%?,,]
[!equal,[!string],y]
        qprint/notify %1%
[!end]
pop
```

This macro uses the !READ pseudomacro to obtain input from the macro user. The input value is placed in the CLI String. The macro then uses the !STRING pseudomacro in a comparison to determine whether or not to print a file.

Notice that the CLI environment level is changed for the duration of the sequence. Moving to a new environment level ensures that the previous contents of the CLI String will not be replaced. The POP command restores the previous environment and any value that was contained in the CLI String.

STRING *Command*

Displays or sets a string value (CLI32 version — CLI16 version precedes).

Format

STRING *[argument] [...]*

The CLI32 STRING command offers access to an unlimited number of strings. There is an unnamed string, which you set and access via STRING commands, and there are multiple named strings which you set and access via STRING/NAME= commands and can access using the !STRING/NAME=pseudomacro.

The name of a named string can be any legal AOS/VS or AOS/VS II filename. The number of characters in a string is unlimited.

For example,

```
) STRING/NAME=EXAMPLE  This is a string. ↵
) STRING/NAME=EXAMPLE ↵
This,is,a,string.
```

The STRING commands create a string named EXAMPLE and stores the text This is a string in it, and then display the string in CLI *normalized* form, which means the CLI inserts commas between arguments. To display the string value without the commas, use the !STRING pseudomacro. For example:

```
) WRITE [!STRING/NAME=EXAMPLE] ↵
This is a string.
```

The argument to the STRING command can be a string, a pseudomacro that expands to a legal string, or a combination of the two. For example, the following command assigns the name of the working directory to variable WORKING_DIR.

```
) STRING/NAME=WORKING_DIR [!DIRECTORY] ↵
```

- No templates.
- Requirement: *Standard.*
- See also: STRING (CLI16 version), !STRING, !SUBSTRING, VAR.

Why Use It?

Use the STRING command to display (and set) the contents of the unnamed and named strings. When you execute a program via the XEQ or PROCESS command, you can use the /S switch to place the program termination message in the unnamed string, and then examine it with the STRING command.

STRING, CLI32 Version (continued)

STRING also lets you enter text and pseudomacro information into a CLI string.

The !STRING pseudomacro lets you use string contents as a command argument and display multiple–word strings without having commas inserted by the CLI.

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=.

/ALL

Displays the values of all the strings 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 strings you have assigned. If you have assigned any named strings, the display includes the value of the unnamed string (even if you have not assigned it). With no switches, /INFO displays the values of all strings at the current environment level. For example,

```
) STRING This is the unnamed string. ↵
) STRING/NAME=STRING_1  This is string_1. ↵
) STRING/INFO ↵
<UNNAMED>=This, is,the,unnamed,string.
STRING_1 = This,is,string_1.
```

For string information on all levels, use /INFO/ALL. The display includes the environment level in parentheses. For example,

```
) STRING/NAME=LEVEL_0 This is a string on level 0. ↵
) PUSH ↵
) STRING/NAME=LEVEL_1 This is a string on level 1. ↵
) STRING/INFO/ALL ↵
LEVEL_0 (0) This is a string on level 0.
LEVEL_1 (1) This is a string on level 1.
```

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 string at all environment levels — again, as with /ALL, with the level shown in parentheses.

STRING, CLI32 Version (continued)

/KILL	Clears (sets to null) a string. If you omit other switches, the CLI clears the unnamed string only. If you include /ALL (/KILL/ALL), the CLI clears all strings *on the current level only*. (This differs from the /INFO/ALL, which displays values on all levels.) If you really want to clear all strings, you can chain to another CLI (CHAIN :CLI).
/LEVEL=n	Sets a string to the value it has in the specified environment level. The CLI assumes the unnamed string unless you specify a string with /NAME=. With the /ALL switch, sets *all* strings to the values they had 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.
/NAME=string–name	(CLI32 only.) Specifies a name for the string. The string–name must have between 1 and 32 characters, each of which is a legal filename character. If you omit this switch, the CLI executes the command on the unnamed string.
/P*[=n]* or /PREVIOUS*[=n]*	Without =*n*, expands to the value a string has at the previous environment level. With =*n*, expands to the value the string has at the specified environment level. The *n* specifies the number of levels above the current level (toward 0). The CLI assumes the unnamed string unless you specify a string with /NAME=. With the /ALL switch, sets all strings to the values they had at the specified environment level.

STRING (CLI32) Examples

Sample dialog with the STRING command follows. Text to the right in parentheses is commentary.

```
) STRING/NAME=AAA ABCDE ↵     (A named string at Level 0.)
) STRING/NAME=AAA ↵
ABCDE

) STRING WWWW ↵               (The unnamed String — uppercase S indicates
                               the unnamed String — at Level 0.)
) STRING/NAME=BB XYZ ↵        (A named string at Level 0.)
```

STRING, CLI32 Version (continued)

```
) STRING/NAME=AAA FGHIJ ↵          (Assign AAA a new value
) STRING/NAME=BB IJIJ ↵             ... and BB a new value.)

) PUSH ↵                           (To Level 2.)
) STRING/NAME=AAA LEVEL2 ↵         (Assign AAA a value here.)
) STRING/NAME=AAA/LEVEL=0 ↵        (Assign AAA its value at Level 0.)
) STRING/NAME=AAA ↵                (Display AAA's value.)
ABCDE

) STRING/NAME=AAA/LEVEL=-1 ↵       (Assign AAA a value at Level 2 minus 1.)
) STRING/NAME=AAA ↵                (Display AAA's value.)
FGHIJ

) STRING/NAME=AAA LEVEL2 ↵         (Assign AAA a value here at Level 2.)
) sTRING FF ↵                      (Assign the unnamed String a value.)
) STRING/INFO ↵                    (Display values at current level.)
<UNNAMED> = FF
AAA = LEVEL2
BB = IJIJ

) STRING/INFO/NAME=AAA ↵           (Display AAA's values at all levels.)
AAA (0) = ABCDE                    (AAA's value at Level 0.)
AAA (1) = FGHIJ                    (AAA's value at Level 1.)
AAA (2) = LEVEL2                   (AAA's value at Level 2.)

) STRING/INFO/ALL ↵                (Display all strings' values at all levels, in
                                    in alphabetical order by string name.)
<UNNAMED> (0) = WWWW               (The unnamed String's value at Level 0.)
<UNNAMED> (2) = FF                 (The unnamed String's value at Level 2.)
AAA (0) = ABCDE                    (AAA's value at Level 0.)
AAA (1) = FGHIJ                    (AAA's value at Level 1.)
AAA (2) = LEVEL2                   (AAA's value at Level 2.)
BB (0) = XYZ                       (BB's value at Level 0.)
BB (1) = IJIJ                      (BB's value at Level 1.)

) STRING/PREVIOUS/ALL ↵            (Set all strings to the values they have at the
                                    previous level — 1.)
) STRING/INFO ↵                    (Display all string values here at level 2.)
<UNNAMED> = WWWW
AAA = FGHIj
BB = IJIJ
```

STRING, CLI32 Version (continued)

) STRING/KILL/ALL ↵	(Clear all strings here at Level 2.)
) STRING/INFO/ALL ↵	(Display all string values at all levels, in alphabetical order by string name.)
<UNNAMED> (0) = WWWW	(Unnamed String value at Level 0.)
<UNNAMED> (2) =	(Unnamed String value at Level 2.)
AAA (0) = ABCDE	(AAA's value at Level 0.)
AAA (1) = FGHIJ	(AAA's value at Level 1.)
AAA (2) =	(AAA's value at Level 2.)
BB (0) = XYZ	(BB's value at Level 0.)
BB (1) = IJIJ	(BB's value at Level 1.)
BB (2) =	(BB's value at Level 2.)
) STRING/NAME=YYY ↵	(String YYY has not received a value, so the CLI assigns it the empty string.)
) WRITE [!LENGTH [!STRING/NAME=YYY]] ↵	
0	(An empty string has 0 characters.)

Two values appear to be missing from the response to the STRING/INFO/ALL command — <UNNAMED> (1) and BB (2). The CLI only displays the string values that you explicitly assigned at the environment level it is displaying. In this case you never explicitly assigned a value to the unnamed String at Level 1 or to named string BB at Level 2.

In other words, if the value of a string at a specific environment level is only a copy of the value of the string at the previous environment level, then the /INFO switch does not display the value at the specific level. String BB at Level 2 is only a copy of its value at Level 1, so the /INFO switch does not display its value at Level 2.

!STRING *Pseudomacro*

Expands to the value of a string.

Format

[!STRING]

This pseudomacro displays the value of a string. In CLI16, it can display the value of the unnamed string only.

In CLI32, it can display the unnamed string and, with the /NAME= switch, values of all strings you have assigned with the STRING/NAME= command. For example, suppose you have previously given the command

```
) STRING/NAME=EXAMPLE This is a string. ↵
```

to both identify string EXAMPLE and assign it a value. You can display the value with either the STRING command or !STRING pseudomacro; for example

```
) WRITE [!STRING/NAME=EXAMPLE] ↵
This is a string.

) STRING/NAME=EXAMPLE ↵
This,is,a,string.
```

With the STRING command, the CLI displays the string with commas added between the arguments.

- No templates.
- No arguments.
- Accepts macro name switches (described later).
- Requirement: *Standard.*
- See also: STRING, !SUBSTRING (CLI32 only), !VAR (CLI32 only), VARn.

Why Use It?

The !STRING pseudomacro displays the contents of a string just as you assigned them, without the commas the CLI would display if you used the STRING command. Use it in macros and commands when you want to display the contents as you assigned them.

With CLI32, you can use !STRING with the /NAME switch to refer to an unlimited number of strings.

!STRING Macro Name Switches

/LEVEL=n — (CLI32 only.) Expands to the value a string has in the specified environment level, n. The CLI assumes the unnamed string unless you specify a string with /NAME=. With the /ALL switch, sets *all* strings to the values they had at the specified environment level.

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.

/NAME=string–name — (CLI32 only.) Specifies a name for the string. String names have between 1 and 32 filename characters. If you omit this switch, the CLI executes the command on the unnamed string.

/P[=n] — Without *=n*, expands to the value a string has at the previous environment level. With *=n* (CLI32 only), expands to the value the string has at the specified environment level. The *n* specifies the number of levels above the current level (toward 0). The CLI assumes the unnamed string unless you specify a string with /NAME=. With the /ALL switch, sets all strings to the values they had at the specified environment level.

/PREVIOUS[=n] — (CLI32 only.) Same as /P.

!STRING Example 1

Sample dialog with the !STRING pseudomacro, using the unnamed string, follows. It shows how to set and display the string values on two environment levels.

```
) STRING The working directory is [!DIRECTORY]. ↵
) WRITE [!STRING] ↵
The working directory is :UDD:JAN:MYDIR.

) PUSH ↵
) DIRECTORY :UTIL ↵
) STRING The working directory is now [!DIRECTORY]. ↵
) WRITE [!STRING] ↵
The working directory is now :UTIL.

) WRITE [!STRING/P] ↵
The working directory is :UDD:JAN:MYDIR.
```

!STRING Example 2

Sample dialog with the !STRING pseudomacro using a named string (CLI32 only) follows. Text to the right in parentheses is commentary.

```
) STRING/NAME=AAA A B C ↵          (A named string at Level 0.)

) STRING/NAME=AAA ↵                (Display its value with STRING.)
A,B,C

) WRITE [!STRING/NAME=AAA] ↵       (Display its value with !STRING.)
A B C

) PUSH ↵                                          (To Level 1)
) STRING/NAME=AAA [!STRING/NAME=AAA]DEFGH ↵  (At Level 1, assign string AAA
                                                   a value.)

) WRITE [!STRING/NAME=AAA] ↵                      (Display its value.)
A B CDEFGH

) WRITE [!STRING/NAME=AAA/PREVIOUS] ↵             (Display its value at the previous
                                                   environment level.)
A B C
```

This example will also work with the unnamed string if you omit the /NAME=AAA switch.

!SUBSTRING *Pseudomacro*

Expands to the designated part of a string (CLI32 only).

Format

[!SUBSTRING character–string]

This pseudomacro, when used with the /ITEM= switch, expands to some or all of the characters in the string of text that is the argument to the pseudomacro. The character–string may not contain the following characters: open and close parentheses (), open and close square brackets [], open and close angle brackets <>, spaces, tabs, commas, and multiple backslashes \. Without the /ITEM switch, the pseudomacro returns null.

In addition to extracting a subset of the character–string argument, you can also:

- pad the expanded string to the left with any character
- pad the expanded string to the right with any character
- reverse the expanded string so that, for example, abcde becomes edcba.

The following example introduces some features of !SUBSTRING. Suppose you type

```
) WRITE [!SUBSTRING/RIGHTFILL=$/ITEM=3–8:15/REVERSE Westborough]↵
$$$$$$$$$orobst
```

The CLI obtains the result $$$$$$$$$orobst as follows.

/RIGHTFILL=$	replaces whatever characters are to the right of /ITEM=3–8 with the fill character $, so the current result is stboro$$$$$$$$$.
/ITEM=3–8	CLI extracts, from the argument, characters 3 through 8 inclusive, which are stboro.
/ITEM=3–8:15	CLI places stboro into the leftmost of 15 characters and pads the characters to the right with spaces, yielding stboro***□□□□□□□□□ (□ indicates a space).
/REVERSE	replaces stboro$$$$$$$$$ with its reverse, so the final result is $$$$$$$$$orobst.

The order of the switches is not important.

- No argument switches.
- Accepts macro name switches (described later).
- Requirement: *Standard.*
- See also: !ARGUMENT, !INDEX, !STRING, !VAR (CLI32 only), !VARn.

!SUBSTRING (continued)

Why Use It?

Use this pseudomacro to extract a substring from a string. This extraction, along with filling and reversing, is helpful as you manipulate strings.

Macro Name Switches

/CONCATENATE

Concatenates (joins) all individual strings that two or more /ITEM= switches specify. Use this switch when you want to concatenate substrings. Without this switch, the individual strings that two or more /ITEM switches specify are separated by spaces. You must use this switch together with one or more /ITEM switches. For example, compare the following commands:

```
) write [!substring/item=2–4/item=6  ABCDEF] ↵
BCD□F

) write [!substring/concatenate/item=2–4/item=6 ABCDEF]↵
BCDF

) write [!substring/item=2–4/item=6/concatenate ABCDEF]↵
BCD□F
```

(The /ITEM=2–4 and /ITEM=6 switches extract characters 2 through 4 and 6 respectively from the text string ABCDEF.)

You can use this switch with multiple ITEM= switches to produce a continuous string of characters. However, enclosing items in parentheses does the same thing as the concatenate switch, but requires you to type fewer characters. For example, both of the following commands (shown in lowercase) yield the same string, ABC.

```
) write  [!substring/item=(1,2,3)  ABCDEF]
ABC

write [!substring/concatenate/item=1/item=2/item=3 ABCDEF]
ABC
```

Don't use the /SEPARATE switch with the /CONCATENATE switch.

!SUBSTRING (continued)

/ITEM=item–list

Specifies the substrings to be extracted from the character–string argument. To specify a single character, use a single integer that identifies the location of the character in the character string. For example, the command

```
) WRITE [!SUBSTRING/ITEM=3 ABCD]
C
```

The command displays *C* because it is the third character in the string.

To specify a range of characters, use an item–list of the form m–n:l*r

where

m — represents the first character in character–string.

n — represents the last character in character–string.

l — represents the number of characters in the extracted substring with, if necessary, left justification and padding with spaces.

r — represents the repeat count.

For example, suppose character–string is cdefghijk. Then

5–7:3*1	specifies substring ghi
3–3:1*4	specifies substring eeee
4–5:2*3	specifies substring fgfgfg

The default values of m, n, l, and r are as follows.

m — 1

n — The number of characters in text_string

l — n–m+1

r — 1

For example, a character string contains cdefghijk. Then

4– specifies substring fghijk

–6 specifies substring cdefgh

– specifies substring cdefghijk

4–6: specifies substring fgh

4–6:5 specifies substring fghbb (b = blank space)

4–6:*2 specifies substring fghfgh

!SUBSTRING (continued)

/ITEM=item–list (continued)

To specify multiple substrings, separate each item–list with commas or blanks *and surround them with parentheses*. For example,

/ITEM=(4,7–9,3–7:6,11) specifies these substrings:

4th character; 7th, 8th, 9th characters; 3rd, 4th, 5th, 6th, 7th characters (and a trailing blank); 11th character.

To keep characters in substrings separate, you can use individual /ITEM= switches (without parenthesizing item_lists) or you can parenthesize item_lists and use the /SEPARATE switch. Use the approach that is most convenient. For example, both of the following commands produce the same characters, A B C. They appear in lowercase to fit.

```
write [!substring/item=1/item=2/item=3 ABCDEF] ↵
A B C

) write  [!substring/separate/item=(1,2,3)  ABCDEF] ↵
A B C
```

/LEFTFILL=fill–chars

Pads the leftmost positions of the result with the designated characters, to the specified length. Use this switch together with the /ITEM switch. For example,

```
) write [!substring/leftfill=abc/item=2–6:9  ASDFJKLQW] ↵
abcaSDFJK
```

The CLI displays *abcaSDFJK* because /ITEM=2–6 extracts SDFJK, /ITEM=2–6:9 creates SDFJK□□□□, and /LEFTFILL=abc both shifts SDFJK to the right and repeatedly fills the characters to the left with abc.

You cannot use any fill character that has special meaning to the CLI, such as a comma, space, or right angle bracket (>). These special characters are listed near the beginning of !SUBSTRING. And, if you specify more than one fill character, then the CLI adds these to the string from left to right until it has reached the length of the string.

/REVERSE

Changes the order of the result by reversing the characters. Use this switch together with the /ITEM switch. This is the last processing step the CLI does regardless of the location of /REVERSE in the list of switches. For example,

```
) write [!substring/leftfill=*/item=1-8/reverse  SUCABA] ↵
ABACUS**
```

The CLI displays *ABACUS* ** because /ITEM=1–8 extracts SUCABA and expands it to SUCABAbb (b = blank space), /LEFTFILL=* both shifts SUBACA to the right and repeatedly fills the characters to left with * to yield **SUCABA, and /REVERSE reverses these characters to create ABACUS**.

/RIGHTFILL=fill–chars

Pads the result on the right with the designated characters to the specified length. Use this switch together with the /ITEM switch. For example,

```
) write [!substring/rightfill=abc/item=2-6:9  ASDFJKLQW] ↵
SDFJKabca
```

The CLI displays *SDFJKabca* because /ITEM=2–6 extracts SDFJK, /ITEM=2–6:9 creates SDFJK□□□□, and /RIGHTFILL=abc repeatedly fills the characters to the right with abc.

You cannot use a fill character that special meaning to the CLI, such as a comma, space, or right angle bracket (>). These special characters are listed near the beginning if !SUBSTRING. And, if you specify more than one fill character, then the CLI adds these to the string from left to right until it has reached the length of the string.

/SEPARATE

Separates with blanks all individual strings that a set of item–lists specifies. Use this switch together with the /ITEM switch. Without this switch, the individual strings that the set of item–lists specifies are not separated. For example,

```
) write [!substring/item=2/item=(4,6)  ABCDEF] ↵
B□DF
```

Compares to

```
) write [!substring/separate/item=2/item=(4,6)  ABCDEF] ↵
B□D□F
```

Don't use this switch with the /CONCATENATE switch.

!SUBSTRING Examples

Next are several commands of the form WRITE [!SUBSTRING/ITEM=] that show how this pseudomacro functions.

1.) WRITE [!SUBSTRING/ITEM=2 abcde] ↵
   b

This !SUBSTRING extracted the second character.

2.) WRITE [!SUBSTRING/ITEM=2/ITEM=1 abc] ↵
   b a

This !SUBSTRING extracted the second character and the first character, and then displayed them, separated by a space.

3.) WRITE [!SUBSTRING/ITEM=(2,1) abc] ↵
   ba

This !SUBSTRING extracted the second character and the first character, and then displayed them (with no separation).

4.) WRITE [!SUBSTRING/SEPARATE/ITEM=(2,1) abc] ↵
   b a

This !SUBSTRING extracted the second character and the first character, and then displayed them, separated by a space.

5.) WRITE [!SUBSTRING/ITEM=1-3 abcdef] ↵
   abc

This !SUBSTRING extracted the first through the third characters.

6.) WRITE [!SUBSTRING/ITEM=-2/REVERSE abc] ↵
   ba

This !SUBSTRING extracted the first through the second characters, and then displayed them in reverse order.

7.) WRITE [!SUBSTRING/RIGHTFILL=*/ITEM=-2:6 abc] ↵
   *ab*****

This !SUBSTRING extracted the first through the second characters, placed them in temporary storage, left justified with four blank spaces to the right, and replaced these blank spaces with the specified fill character, asterisk (*).

8.) WRITE [!SUBSTRING/RIGHTFILL=cd/ITEM=-2:6/REVERSE abc] ↵
   dcdcba

This !SUBSTRING extracted the first through the second characters, placed them in temporary storage, left justified with four blank spaces to the right, replaced these blank spaces with the fill characters cd for a temporary result of abcdcd, and reversed this temporary result to obtain the final result.

9.) WRITE [!SUBSTRING/ITEM=2–7:3*2 abcdefg] ↵
bcdbcd

This !SUBSTRING extracted the second through the seventh characters (bcde), placed as many as would fit into a 3–character area (bcd), and repeated this result so that it occurred twice.

10.) WRITE [!SUBSTRING/ITEM=(1–3,4–6) abcdef] ↵
abcdef

This !SUBSTRING extracted the first through third and fourth through sixth characters, and display them without a separator.

11.) WRITE [!SUBSTRING/SEPARATE/ITEM=(1–3,4–6) abcdef] ↵
abc def

This !SUBSTRING extracted the first through third and the fourth through sixth characters with a space to separate.

12.) WRITE [!SUBSTRING/ITEM=1–3/ITEM=4–6 abcdef] ↵
abc def

This !SUBSTRING used a different form to extract the first through third characters and then the fourth through sixth characters, with a space to separate.

13.) WRITE [!SUBSTRING/ITEM=6/ITEM=(1–3,4–6) abcdef] ↵
f abcdef

This !SUBSTRING extracted the sixth character, and extracted the first through third and fourth through sixth characters. Then it displayed these two character strings with a separating space.

14.) WRITE [!SUBSTRING/CONCATENATE/ITEM=1–3/ITEM=4–6 abcdef] ↵
abcdef

This !SUBSTRING pseudomacro extracted the first through third characters and then the fourth through sixth characters. /CONCATENATE told the CLI not to take the default action of inserting a space to separate.

15.) WRITE [!SUBSTRING/LEFTFILL=1234/ITEM=–:10 abc] ↵
1234123abc

This !SUBSTRING pseudomacro extracted all characters and placed them in a 10–character temporary storage area as abc␣␣␣␣␣␣␣ (seven spaces). Then it right–justified abc and repeatedly filled the leftmost seven spaces with 1234.

SUPERPROCESS *Command*

Displays or sets the Superprocess mode.

Format

SUPERPROCESS [*ON* | *OFF*]

This command either reports whether the Superprocess privilege is currently turned on or off, or, if you are allowed the Superprocess privilege, lets you turn it 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.

Your user profile, as created by the system manager, determines whether or not you have the Superprocess privilege. If you have it, you can turn it on or off as needed.

For CLI16, the Superprocess prompt is +) or, if Superuser mode is also turned on, #). For CLI32, the Superprocess prompt is *Sp*). (The CLI32 prompt may also include the characters *Su* and/or *Sm*, if Superuser and/or System Manager mode are also on.)

- No templates.
- No argument switches.
- Requirement: *Standard* to display; *Superprocess* to turn on.
- See also: SUPERUSER, PRIVILEGE SYSTEMMANAGER (CLI32 only).

Why Use It?

Use the SUPERPROCESS command to turn Superprocess mode on or off, provided you have the Superprocess privilege. You'll need this privilege if you have to control processes that are outside your own process tree.

The Superprocess privilege is necessary for certain operations using the BLOCK, BREAKFILE, PRIORITY, PRTYPE, TERMINATE, and UNBLOCK commands.

Even if you don't have Superprocess privilege, you can still display information about all processes by using the WHO and RUNTIME commands.

A Word of Caution

The Superprocess privilege is a powerful tool, which should be exercised with caution. Because this privilege overrides certain safeguards within the system, you should be careful when performing any operation that could interfere with another process.

SUPERPROCESS 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=.

/LEVEL=n — (CLI32 only.) Sets Superprocess mode to the setting it has in 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 Superprocess mode to the setting it has at the previous environment level. With *=n* (CLI32 only), sets Superprocess mode to the setting it has at the specified environment level. The *n* specifies the number of levels above the current level (toward 0).

/PREVIOUS[=n] — (CLI32 only.) Same as /P.

SUPERPROCESS Example 1

```
) TREE ↵
PID:  32, Father:   3, Sons: 47
) TERMINATE 45 ↵
Error: Attempt to access process not in hierarchy

) SUPERPROCESS ON ↵
+) TERMINATE 45 ↵              (The CLI16 Superprocess prompt prefix is +)
+) SUPERPROCESS  OFF ↵
)
```

The SUPERPROCESS command allowed this user to terminate PID 45, which is outside the user's process tree.

SUPERPROCESS Example 2

```
) SUPERPROCESS ON ↵
Sp) DELETE/2=IGNORE PROCESSDUMP ↵     (CLI32 prompt prefix is Sp)

Sp) BREAKFILE/ALL/FILENAME=PROCESSDUMP 37 ↵
Sp) TERMINATE 37 ↵
Sp) SUPERPROCESS  OFF ↵
)
```

The SUPERPROCESS command enables this user to create a break file format and terminate a process that was malfunctioning or deadlocked. The information in freshly created file PROCESSDUMP can then be submitted with a Software Trouble Report (STR).

SUPERUSER *Command*

Displays or sets the Superuser mode.

Format

SUPERUSER [*ON* / *OFF*]

This command either reports whether the Superuser privilege is currently turned on or off, or, if you are allowed Superuser privilege, lets you turn it on or off.

The Superuser privilege, when turned on, allows you to gain access to any file on the system (overriding limitations set by access control lists). Also, certain CLI commands provide functions that are available only when you have the Superuser privilege turned on.

Your user profile, which is built by the system manager, determines whether or not you have the Superuser privilege. If you have it, you can turn it on or off as needed.

For CLI16, the Superuser prompt is *) or, if Superprocess mode is also turned on, it is #). For CLI32, the Superuser prompt is *Su*). (The CLI32 prompt may also include the prefix characters *Sp* and/or *Sm*, if Superprocess and/or System Manager mode are also on.)

- No templates.
- No argument switches.
- Requirement: *Standard* to display; *Superuser* to turn on.
- See also: SUPERPROCESS, PRIVILEGE SYSTEMMANAGER (CLI32 only).

Why Use It?

Use the SUPERUSER command to turn Superuser mode on or off, provided you have the Superuser privilege.

You may want Superuser on to read or modify a file that you could not otherwise access. For example, you may need to place a new program in a common directory, which grants you only Read and Execute access. With Superuser mode on, you can write in that directory. If you want to be able to create files in the directory without using Superuser mode, you can change the directory ACL to give yourself Append or Write access.

A Word of Caution

The Superuser privilege is a powerful tool, which should be exercised with caution. Because this privilege overrides certain safeguards within the system, you should be careful when performing any operation that could change or delete information.

SUPERUSER 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=.

/LEVEL=n — (CLI32 only.) Sets Superuser mode to the setting it has in 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 Superuser mode to the setting it has at the previous environment level. With *=n* (CLI32 only), sets Superuser mode to the setting it has at the specified environment level. The *n* specifies the number of levels above the current level (toward 0).

/PREVIOUS[=n] — (CLI32 only.) Same as /P.

SUPERUSER Example 1

```
) DIRECTORY :
) LOAD_II/V @MTD0:0
```

Warning: Write or append access is required, File :UTIL:SORT32
Warning: Write or append access is required, File REPORT.01

```
) SUPERUSER ON
*) LOAD_II/V @MTD0:0                    (The CLI32 Superuser prompt is Su)

:UTIL:SORT32
REPORT.01
*) SUPERUSER OFF
)
```

The first two commands try to load a file from tape into the root directory, but error messages indicate that the needed access types are lacking. Turning on the Superuser privilege allows the person to override the access control list of the root directory.

SUPERUSER Example 2

```
) SUPERUSER ON ↵
Su) MOVE/V :UDD:TERRY  STATUS_REPORT.CHRIS ↵    (The CLI16 Superuser
                                                 prompt is *)
STATUS_REPORT.CHRIS
Su) SUPERUSER OFF ↵
)
```

User CHRIS turns on Superuser to move a file into user TERRY's directory, which does not grant CHRIS access.

SUPERUSER Example 3

```
) PUSH ↵                          (Push down to Level 1.)

) SUPERUSER ON ↵                  (Turn Superuser mode on.)

*) TYPE :MYSYS.CSF ↵              (CLI16 Superuser prompt is *. Type a system
                                   specification file in directory :SYSGEN.)

*) POP ↵                          (Return to previous level, restoring its
)                                  Superuser setting — Off.)
```

SYSID *Command*

Displays or sets the unique system identifier.

Format

SYSID *[identifier]*

This command either displays the current system identifier or lets the system operator assign a new identifier for the system. The identifier string can consist of as many as 32 characters.

The system prints the system identifier on the header pages of print jobs. Also, the system displays the identifier on user terminals EXEC has enabled, *unless* a banner has been defined in the banner file, :UTIL:LOGON.BANNER.SCREEN.

The system identifier is not necessarily the same as the hostname or default operating system pathname.

- No templates.
- No argument switches.
- Requirement: *Standard* to display; *PID 2 or System Manager* to set.
- See also: CPUID, SYSINFO, HOST.

Why Use It?

Use the SYSID command to display the unique system identifier. If you frequently log on remote terminals, you can use this command to show which system you are currently using.

If you are the PID 2 or if you have the System Manager privilege, you can use this command to set or change the system identifier. Your system's UP.CLI macro can include a SYSID command to set the identifer.

Some systems include the current operating system revision number as part of the system identifier. You could use the SYSID command to update the system identifier after you install a new revision. (The default system identifier is the revision number of the operating system.)

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=.

SYSID Examples

) SYSID ↵ *PLINY_7.69*	(This command displays the unique system identifier, PLINY_7.69.)
) SYSID PLINY ↵	(This command, executed by the system operator (PID 2), changes the system identifier to PLINY.)

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*).

SYSLOG (continued)

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.

TAR_VS (continued)

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.

TAR_VS Example 2

```
) TAR_VS/CREATE/FILENAME=@MTJ0:0/VERBOSE [!FILENAMES +] ↵
a- MY_FILE 21 blocks
a- X_FILE 44 blocks
a- JUNE_DATA 3144 blocks
.
)
```

This example shows TAR_VS dumping all files in the working directory. As with the previous example, the /VERBOSE switch has the utility display file information as it works.

TAR_VS Example 3

```
) TAR_VS/TELL/VERBOSE/FILENAME=@MTJ0:0 ↵


.
(Displays filenames on tape without loading them)
.
) TAR_VS/XTRACT/VERBOSE/FILENAME=@MTJ0:0 ↵


.
(Displays filenames loaded)
)
```

This first command 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 tar –create.

TERMINATE *Command*

Terminates a process.

Format

TERMINATE { process–ID / processname / username:processname } *[...]*

This command terminates one or more processes, removing them from the process hierarchy. If you have turned on Superprocess or are working from the master CLI (PID 2), you can terminate any process.

You must supply a process ID or a process name. You can use a process name only if the process was created with the PROCESS command and /NAME=name switch.

- No templates.
- No argument switches.
- Requirement: *Standard* to terminate the current process or its son(s); *Superprocess* to terminate any process.
- See also: PROCESS, SUPERPROCESS, RUNTIME, BLOCK.

Why Use It?

Use the TERMINATE command to stop a process prior to its normal termination. For example, you can use this command to stop a process that is caught looping and consuming a lot of CPU time.

(The RUNTIME command can tell you how much CPU time and I/O a process is doing. Use these as an indication of process behavior.)

A Word of Caution

Terminating a process abnormally (as with TERMINATE) leaves any files the process was writing in a undetermined state. It may eliminate all useful work the process has done. If you have doubts about whether you want to terminate a process, you can block it (BLOCK command) and decide at leisure.

TERMINATE 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=.

/BREAKFILE*[=pathname]* Creates a break file of the terminated process. If you do not supply a pathname for the break file, the system assigns the break file a name, and places the file in your working directory. The format for the filename is ?pid.time.BRK.

TERMINATE Example 1

```
) PROCESS SMITH:PROGA ↵
PID  17
...
...
) TERMINATE 17 ↵
```

The first command creates a subordinate process, which runs as PID 17. Later, the TERMINATE command terminates that process. In this example, the following commands are equivalent to the last command:

```
TERMINATE SMITH:PROGA
 and
TERMINATE PROGA
```

TERMINATE Example 2

```
) TREE ↵
PID:   22, Father:    3, Sons:  33

) TERMINATE 34 ↵
Warning:  Attempt to access process not in hierarchy
```

This TERMINATE command was unsuccessful because the target process was not the current CLI process nor its son.

TERMINATE Example 3

```
) SUPERPROCESS  ON ↵
+) TERMINATE  ARIES ↵
+) SUPERPROCESS  OFF ↵
)
```

The TERMINATE command terminates the process known as ARIES. The Superprocess privilege lets you terminate a process that is not a son of the process that issues the command.

TERMINATE Example 4

This user is running the CLI as process 27 (PID 27), and has created input and output files and another process with these commands:

```
) CREATE/I  INFILE ↵
)) RP MEMO ↵
)) ) ↵
) CREATE  OUTFILE ↵
) PROCESS/INPUT=INFILE/OUTPUT=OUTFILE  :CLI ↵
PID: 34
```

The process runs concurrently with its parent and executes the RP macro that takes argument MEMO. (Works only with the *Create without block* privilege.) To terminate the process, the uses types

```
) TERMINATE  34 ↵
```

TIME *Command*

Displays or sets the system time.

Format

TIME *[hours [[:]minutes [[:]seconds]]]*

This command either displays or sets the current system time.

- No templates.
- No argument switches.
- Requirement: *Standard* to display; *PID 2* or *System Manager* to set.
- See also: !TIME, DATE, !DATE.

When displaying the time, the system uses the format hh:mm:ss. When setting the time, you must specify hours using a 24–hour clock. You can also specify either minutes (0 through 59), or minutes and seconds (0 through 59). Use a colon or space to separate the hours, minutes, and seconds. (See the example below.)

Why Use It?

This command is useful if you want to know what time it is. You can include the TIME command as part of the CLI prompt (see the PROMPT command) so that the CLI shows the current time whenever it displays its prompt.

If you are running the master CLI (PID 2) or if you have the System Manager privilege turned on, you can reset the system time; for example, if this time is wrong, or to change between standard and daylight savings time.

NOTE: If you set the time forward while the multiuser environment (EXEC) is running, you may confuse EXEC as it monitors user log–on times. If this happens, EXEC may display the error messages *Negative time encountered* and *Internal consistency error in EXEC.* If you see this message, shut down the multiuser environment; then restart 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 switch /STR=.

TIME Example

```
) TIME ↵                                  (Display the current time.)
13:57:46
) PRIVILEGE SYSTEMMANAGER ON ↵            (Only PID 2 or a process with System
Sm)                                        Manager privilege can set the time.)
Sm) TIME 13 18 ↵                          (Set the time to 1:18 p.m.)
Sm) TIME ↵                                (Verify the change. )
13:18:03
```

!TIME *Pseudomacro*

Expands to the system time or converts time to an integer value.

Format

[!TIME [*/NUMERIC [hh[:mm[:ss]]]*[1] | *integer*[1]]]

1 CLI32 only

This pseudomacro returns the current system time, using a 24–hour clock, in hh:mm:ss format — or, with CLI32, converts time to an integer value and vice–versa.

To convert a time value (format hh*[:mm[:ss]]*) to the integer number of seconds after midnight, use the form [!TIME/NUMERIC ...]. If you omit a time value, the system converts the current time. To convert an integer (range 0 through 86399) to a time value, use the form [!TIME integer].

- Argument allowed in CLI32 only.
- Macro name switch in CLI32 only.
- Requirement: *Standard.*
- See also: TIME, !DATE, DATE.

Why Use It?

This pseudomacro lets you include the current time within a statement. For example, a macro could use the !TIME pseudomacro in a WRITE statement to report the time that an action took place. With CLI32, you can use the !TIME and !TIME/NUMERIC to measure the passage of time.

Macro Name Switch

/NUMERIC — (CLI32 only.) Displays the current time as a number of seconds since midnight; or if you include a time, converts the time you specify into seconds past midnight.

!TIME Example 1

```
) WRITE The time is [!TIME].
```
The time is 08:36:35.

This command displays a message that reports the current system time.

!TIME Example 2

(within a macro)

```
write Time completed:,,[!time]
```

The WRITE command displays a message showing the time at which an action was completed.

!TIME Example 3

A macro contains the following commands.

```
write Starting sort at [!time]
var/name=time_started [!time/numeric]
xeq  my_sortprog  july.info
var/name=time_ended [!time/numeric]
write Sort ended at [!time]
write The elapsed time for the sort was &
[!usubtract [!var/name=time_ended] [!var/name=time_started]] seconds.
```

For CLI32, these commands use !TIME to determine how long a program took to run. The display as the macro ran might be

Starting sort at 16:24:20
Sort ended at 16:45:23
The elapsed time for the sort was 1263 seconds.

TRACE *Command*

Displays or sets the current trace mode.

Format

TRACE

The TRACE command either displays the current trace mode or lets you set the trace mode. You can turn tracing on or off for commands, macros, pseudomacros, or a combination of these.

The tracing feature lets you observe how the CLI processes commands, macros, and pseudomacros. When trace mode is on, the CLI displays the actual command line before executing it.

CLI16 uses the following notations in the trace output:

Symbol	Signifies
***	A CLI command
###	A macro
+++	A pseudomacro

CLI32 displays trace output as follows:

```
v-------------------------------------v
xxx                                    (The xxx is the trace message.)
^-------------------------------------^
```

- No arguments.
- Requirement: *Standard.*

Why Use It?

Use the TRACE command to determine if any tracing is in effect, or to verify the series of commands executed in a macro.

TRACE can be particularly helpful in debugging a macro that has complex conditionals, or in tracing execution when a macro call fails.

TRACE features and display with CLI32 make it much easier to use than those of CLI16, so we recommend you use CLI32 for tracing if possible.

TRACE 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.

/ALL	(CLI32 only.) Turns trace on for all pertinent items: commands, files, pseudomacros, and macros. This is the equivalent of using switches /COMMAND, /CONDITIONAL, /FILES, /MACRO, /PSEUDO, and REPLACEMENT. The only reason you might not want to use this switch is that it gives you too much information for the task you're approaching; in such cases, be selective.
/COMMAND	Turns tracing on or off for commands (does not affect tracing of files, macros, or pseudomacros). You can include the /ON or /OFF switch. If you omit /ON or /OFF, this switch turns command tracing on.
/CONDITIONAL	(CLI32 only.) Turns tracing on or off for text included in conditional pseudomacros. You can include the /ON or /OFF switch. When this is on, for every conditional pseudomacro the CLI displays information about included and excluded text. Tracing conditionals can help you understand execution in macros that use many and/or elaborate conditional pseudomacros.
/FILES	(CLI32 only.) Turns tracing on or off for files (does not affect tracing of commands, files, or pseudomacros). You can include the /ON or /OFF switch. If you omit /ON or /OFF, this switch turns tracing on. With CLI32, you can also use MACRO to display macro pathnames.
/KILL	Turns all tracing off. This switch must be used alone.
/LOG	Sends trace output to the current log file.
/MACRO	Turns tracing on or off to display the names of macro files the CLI enters and leaves. You can include the /ON or /OFF switch. Tracing macros tells you the pathnames of all macros used.
/OFF	Turns off the types of tracing specified by the switch(es) that follow.
/ON	Turns on the types of tracing specified by the switch(es) that follow.

TRACE (continued)

/LEVEL=n — (CLI32 only.) Sets the trace mode to the one used in 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 trace mode to the setting used in the previous environment. (Omit the pathname arguments.) With *=n* (CLI32 only), sets the trace mode to the one used in the previous CLI environment level. The *n* specifies the number of levels above the current level (toward 0).

/PREVIOUS*[=n]* — (CLI32 only.) Same as /P.

/PSEUDO — Turns tracing on or off for pseudomacros (does not affect tracing of commands, files, or macros). You can include the /ON or /OFF switch. If you omit /ON or /OFF, this switch turns pseudomacro tracing on.

/REPLACEMENT — (CLI32 only.) Turns tracing on or off for replacement of dummy arguments with actual arguments. You can include the /ON or /OFF switch. When this is on, for every dummy argument, the CLI displays the original dummy argument and the actual argument that replaces it. Tracing dummy argument replacement can help you understand execution in macros that use dummy arguments.

TRACE Example, CLI16

This example shows TRACE with CLI16. The CLI32 example follows this one.

The macro BEEP.CLI contains

```
[!equal %1%,BEEP]
      write Hello [!ascii 207]
[!else]
      write Argument 1 was not BEEP.
[!end]
```

TRACE Example, CLI16 (continued)

The following commands show tracing with the BEEP.CLI macro:

) TRACE ↵	(Shows the current trace modes.) (All tracing is off.)
) TRACE/COMMAND/MACRO/PSEUDO↵	(Turns on tracing for all modes.)
) BEEP BEEP ↵	(Executes the macro BEEP.CLI with argument BEEP.)
###*[beep,BEEP]*	(The system displays the macro.)
+++*[!equal,BEEP,BEEP]*	(The system displays the pseudomacro.) (The +++ indicates a pseudomacro, not a command or macro.)
*** *write,Hello*	(The system displays the WRITE command.) (The *** indicates a command, not a macro or pseudomacro.)
Hello<beep>	(The terminal beeps.)
) TRACE/K ↵	(Ends all tracing.)

The first command checks tracing. The next command turns on tracing for commands, macros, and pseudomacros. The third command executes the macro. The system shows the following trace information: an echo of the macro, and then an echo of the pseudomacro and command with the macro. The last line (the *Hello* and the beep) is the output from the macro.

TRACE Example, CLI32

This example shows TRACE with CLI32. (The CLI16 example precedes.) As with the CLI16 example, the macro BEEP.CLI contains

```
[!equal %1%,BEEP]
        write Hello [!ascii 207]
[!else]
        write Argument 1 was not BEEP.
[!end]
```

The following commands show tracing with the BEEP.CLI macro:

) TRACE ↵	(Shows the current trace modes.) (All tracing is off.)
) TRACE/ALL ↵	(Turns on tracing for all modes.)
	(CLI traces TRACE command.)

```
v-------------------------------v
Command: TRACE/ALL
^-------------------------------^
```

TRACE Example, CLI32 (continued)

```
) BEEP BEEP ↵                                  (Executes the macro BEEP.CLI
v————————————————v                             with argument BEEP.)
Macro name: beep.CLI                           (CLI displays the macro name,
Expands to: [!equal %1%,BEEP]                  and expansion of the conditional,
write [!ascii 207]                             and the rest of the macro.)
[!else]
write Argument 1 was not BEEP.
[!end]
v————————————————————————v
Dummy argument: %1%                            (CLI explains replacement of the
Is replaced by: BEEP                           dummy argument.)
^————————————————————————^
v————————————————————————v                    (CLI explains macro filenames.)
Entering macro: :UDD1:JAN:CLI_MANUAL:BEEP.CLI
^————————————————————————^
v————————————————————————v                    (CLI explains pseudomacro !EQUAL.)
Pseudo-macro: [!equal,BEEP,BEEP]
Expands to: "TRUE"
^————————————————————————^
v————————————————————————v                    (CLI explains conditional !EQUAL.)
Included text:      write [!ascii 207]
^————————————————————————^
v————————————————————————v
Excluded text:
     write Argument 1 was not BEEP.
^————————————————————————^
^v———————————————————————v                    (CLI explains pseudomacro !ASCII.)
Pseudo-macro: [!ascii,207]
Expands to: ^G
^————————————————————————^
v————————————————————————  v                  (CLI explains command WRITE.)
v————————————————————————————v
Command: write, Hello,^G
^————————————————————————^


Hello <beep>                                   (Terminal beeps.)

v————————————————————————  v                  (CLI explains macro filenames.)
Leaving macro: :UDD1:JAN:CLI:BEEP.CLI
^————————————————————————^
) TRACE/K ↵                                    (Ends all tracing.)
```

The first command checks tracing. The next command turns on tracing for all modes. The third command executes the macro. The system shows the following trace information: an echo of the macro, the replacement of dummy argument, files (macro pathname), conditional, pseudomacro, command (WRITE), and file (macro pathname).

TREE *Command*

Displays a process' family tree.

Format

TREE [*[hostname]:process-ID* | *processname* | *username:processname*] *[...]*

This command reports the father and son process IDs for either the current CLI or a specified process.

You can supply a process ID or a process name. If you supply a simple process name, the CLI assumes your username. You can use either process name form, but only if the process was created with the PROCESS command and /NAME=name switch.

If your system is on a XODIAC/XTS network, you can specify a hostname to obtain information about processes on other systems.

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

Why Use It?

The TREE command lets you find out which processes are directly related to a specific process. Before terminating a process, you may want to know if that process has any sons. You also may be interested in knowing which process started (is the father of) a particular process. You can use the WHO macro to learn the username associated with a PID.

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.

TREE Examples

) TREE 17 ↵
PID: 17, Father: 3, Sons: 8 12 13

This command displays the process tree for PID 17. PID 3 is the father process (EXEC), and the process has three sons.

) TREE ↵
PID: 6, Father: 3, Sons:

This command (without arguments) displays the process tree for the current CLI. Again, its father is PID 3 (EXEC); it has no sons.

TYPE *Command*

Displays the contents of a file.

Format

TYPE pathname *[...]*

This command displays the contents of the specified file.

- Accepts templates.
- No argument switches.
- Requirement: *Standard* for a file to which you have Read access and to whose parent directory you have Execute access; *Superuser* for any file.
- See also: QPRINT, DISPLAY, BROWSE.

Why Use It?

Use the TYPE command to display the contents of a text file on your terminal.

Helpful Hints

If the contents of a file exceed the screen size, you can pause the display in any of the following ways:

- Use the HOLD key (not available on all keyboards) to hold and release the display; or
- Press CTRL–S to suspend the display, and CTRL–Q to resume it; or
- Turn on page mode to display one screen at a time, then use CTRL–Q to resume display (use CHARACTERISTICS/PM as explained in the CHARACTERISTICS command.)

If you accidentally type a nontext file, meaningless characters will appear on the screen, often accompanied by beeping. To return your terminal to its normal state, press the following sequence one or more times, as necessary, when output stops:

Break sequence (press CMD and BREAK/ESC keys together); CTRL–S; CTRL–C; CTRL–A; CTRL–Q.

If you want to clear your screen and/or return display to bright mode, press the ERASE PAGE key.

TYPE 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=. Note that /STR= does not work as usual with the TYPE and COPY commands — it returns null .

/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–mmm–yy:hh:mm:ss), date (dd–mmm–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–mmm–yy:hh:mm:ss), date (dd–mmm–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 typed.

/SORT — (CLI32 only.) Sorts alphabetically the filenames this command processes. To see the names, also use /V.

/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

```
TYPE/TYPE=\CPD #:PROJ–.SR
```

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

```
TYPE/TRAVERSE=\CPD #:PROJ–.SR
```

/TYPE=typecode — (CLI32 only.) Specifies one or more types of files to process. All values of type codes appear in Table 2–8. Pertinent types include TXT, UDF, and UNX.

TYPE (continued)

/V	Displays the filename and record type of the file before displaying its contents. For files with fixed–length records, this switch shows the record length. Useful with /TYPE= and/or /AFTER and /BEFORE switches.
/VERIFY	(CLI32 only.) Same as /V.

TYPE Examples

```
) TYPE  MYFILE ↵
...
...
```

This command displays the contents of MYFILE on the screen.

```
) TYPE/L=MYCOPY  :UDD:COMMON:STATUS_REPORT.JULY ↵
```

This command sends the display of the specified file to a list file called MYCOPY in the working directory.

!UADD *Pseudomacro*

Expands to the sum of integers.

Format

[!UADD [*integer1 integer2* / *integer [...]*[1]]]

[1] CLI32 only.

This pseudomacro adds the values of specified integers and returns the sum. Integers must have unsigned, decimal values in the range 0 through 4,294,967,295. With CLI16, you can add *two* integers; with CLI32, you can add as many as you want (up to the limit on line length).

Without integer arguments, the pseudomacro returns the value 0.

If the sum of integers is greater than 4,294,967,295, the pseudomacro returns a value equal to the actual sum modulo 4,294,967,296; that is, the remainder produced by dividing the sum by 4,294,967,296.

- No templates.
- No argument switches.
- No macro name switches.
- Requirement: *Standard*.
- See also: !UDIVIDE, !UMULTIPLY, !USUBTRACT, !UMODULO, VAR and !VAR (CLI32 only), VARn and !VARn.

Why Use It?

Use the !UADD pseudomacro to add integer values and display the result or use it in a test or calculation. As integer values, you can use the VARn and VAR pseudomacros.

!UADD Example 1

```
) WRITE [!UADD 5 6] ↵          (Adds 5 and 6 and displays the result.)
11
```

This command

!UADD Example 2

```
) WRITE [!UADD 5 6 7 8] ↵      (Adds 5, 6, 7, and 8 and displays the result.)
26
```

!UADD Example 3

The following macro uses the !SIZE pseudomacro to evaluate the byte count of two files, and then uses !UADD to add the sizes. The macro then displays the result on the screen.

```
comment Macro ADD_SIZES -- adds byte count of two files.
comment Calling sequence is    macro-name  file1  file2
var0 [!size %1%]
var1 [!size %2%]
write The combined length of files %1% and %2% is
write [!uadd [!var0] [!var1]] bytes.
```

One might use the macro as follows.

```
) ADD_SIZES MYFILE YOURFILE ↵
```

The combined length of files MYFILE and YOURFILE is 2257 bytes.

As another example, see the CALCULATOR macro in Chapter 4.

!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.

!UEQ *Pseudomacro*

Tests two unsigned decimal integer arguments for equality.

Format

```
[!UEQ integer1 integer2]
        ...
        ...
[ [!ELSE] ]
        ...
        ...
[!END]
```

This pseudomacro compares two arguments, each of which must be an unsigned decimal integer value in the range 0 through 4,294,967,295.

If the arguments are equal, the CLI executes the statements that follow the !UEQ statement until the corresponding !ELSE or !END.

Otherwise, the CLI ignores the statements up to the corresponding !ELSE or !END statement. If the sequence includes an !ELSE statement, the CLI executes the statements that follow !ELSE up to the corresponding !END.

- No templates.
- No argument switches.
- No macro name switches.
- Requirement: *Standard*.
- See also: !ELSE, !END, !EQUAL, !NEQUAL, !UGE, !UGT, !ULE, !ULT, !UNE, !UMAXIMUM, !UMINIMUM, VAR and !VAR (CLI32 only), VARn and !VARn.

Why Use It?

Use the !UEQ pseudomacro to test two integer values for equality, and use this information to determine whether or not a sequence of statements should be performed. If you want to compare string values, use !EQUAL instead.

!UEQ Example

(within a macro)

```
[!ueq, [!pid], 2]
        xeq  manage.pr %-%
[!else]
        write Only PID 2 can perform this action.
[!end]
```

The first statement checks the value of the current process ID. If it is equal to 2, the macro executes a program called MANAGE.PR, and passes the macro arguments to the program. If the process that called the macro is not PID 2, the macro writes a message to the screen.

!UGE *Pseudomacro*

Tests the first argument to see if it is greater than or equal to the second.

Format

```
[!UGE integer1 integer2]
        ...
        ...
[ [!ELSE] ]
        ...
        ...
[!END]
```

This pseudomacro compares two arguments, each of which must be an unsigned decimal integer value in the range 0 through 4,294,967,295.

If integer1 is greater than or equal to integer2, the CLI executes the statements that follow the !UGE statement until the corresponding !ELSE or !END statement.

Otherwise, the CLI ignores the statements up to the corresponding !ELSE or !END statement. If the sequence includes an !ELSE statement, the CLI executes the statements that follow !ELSE up to the corresponding !END statement.

- No templates.
- No argument switches.
- No macro name switches.
- Requirement: *Standard*.
- See also: !ELSE, !END, !UEQ, !UGT, !ULE, !ULT, !UNE, !UMAXIMUM, !UMINIMUM, VAR and !VAR (CLI32 only), VARn and !VARn.

Why Use It?

Use the !UGE pseudomacro to compare two integer values, and use this information to determine whether or not a sequence of statements should be performed.

!UGE Example

The macro DIFF.CLI contains the following lines:

```
[!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.

!UGT *Pseudomacro*

Tests the first argument to see if it is greater than the second.

Format

```
[!UGT integer1 integer2]
        ...
        ...
[ [!ELSE] ]
        ...
        ...
[!END]
```

This pseudomacro compares two arguments, each of which must be an unsigned decimal integer value in the range 0 through 4,294,967,295.

If integer1 is greater than integer2, the CLI executes the statements that follow the !UGE statement until the corresponding !ELSE or !END statement.

Otherwise, the CLI ignores the statements up to the corresponding !ELSE or !END statement. If the sequence includes an !ELSE statement, the CLI executes the statements that follow !ELSE up to the corresponding !END statement.

- No templates.
- No argument switches.
- No macro name switches.
- Requirement: *Standard*.
- See also: !ELSE, !END, !UEQ, !UGE, !ULE, !ULT, !UNE, !UMAXIMUM, !UMINIMUM, VAR and !VAR (CLI32 only), VARn and !VARn.

Why Use It?

Use the !UGT pseudomacro to compare two integer values, and use this information to determine whether or not a sequence of statements should be performed.

!UGT Example 1

The macro POPBACK.CLI contains the following lines:

```
comment This is macro POPBACK.CLI.
[!ugt, [!level], 0]
        pop
        %0%
[!end]
```

This macro checks the current CLI environment level. If the level number is greater than 0, the macro moves down a level and calls the macro again. When the level number is equal to 0, the macro does nothing.

!UGT Example 2

The macro QUOTIENT.CLI contains the following commands:

```
var0 [!read Enter dividend: ,,,]
var1 [!read Divide it by: ,,,]
[!ugt,[!var1],0]
        write The quotient is [!udivide,[!var0],[!var1]].
[!else]
        write You cannot divide by 0.
[!end]
```

The first statement prompts for the dividend value, assigning it to the CLI variable VAR0. The second statement requests the divisor value, assigning it to VAR1. The macro then uses the !UGT pseudomacro to ensure that the divisor is greater than 0. Sample dialog is

```
) QUOTIENT ↵
Enter dividend:  36 ↵
Divide it by:  7 ↵
The quotient is 5.
```

!ULE *Pseudomacro*

Tests the first argument to see if it is less than or equal to the second.

Format

[!ULE integer1 integer2]

...

...

[[!ELSE]]

...

...

[!END]

This pseudomacro compares two arguments, each of which must be an unsigned decimal integer value in the range 0 through 4,294,967,295.

If integer1 is less than or equal to integer2, the CLI executes the statements that follow !UGE until the corresponding !ELSE or !END.

Otherwise, the CLI ignores the statements up to the corresponding !ELSE or !END. If the sequence includes an !ELSE statement, the CLI executes the statements that follow !ELSE up to the corresponding !END.

- No templates.
- No argument switches.
- No macro name switches.
- Requirement: *Standard.*
- See also: !ELSE, !END, !UEQ, !UGE, !UGT, !ULT, !UNE, !UMAXIMUM, !UMINIMUM, !UNE, VAR and !VAR (CLI32 only), VARn and !VARn.

Why Use It?

Use the !ULE pseudomacro to perform a sequence of statements when an integer value is less than or equal to a given value. You can use !ULE to ensure that a value does not exceed a maximum limit.

!ULE Example

The macro SQUARE.CLI takes an integer argument, squares it, and displays the result. The macro uses !ULE to ensure that the argument does not exceed the maximum value that the CLI is able to square.

```
comment This is macro SQUARE.CLI.
comment The largest possible result is 4294967295.
comment So, the largest possible argument is 65535.
[!ule %1% 65535]
        write %1% squared is [!umultiply %1% %1%].
[!else]
        write You can't square a number larger than 65535.
[!end]
```

If the argument is less than or equal to 65535, the macro multiplies the value by itself and displays the result. If the value is greater than 65535, the macro displays an error message. For example,

```
) SQUARE 567 ↵
567 squared is 321489.

) SQUARE 999999 ↵
You can't square a number larger than 65535.
```

!ULT *Pseudomacro*

Tests the first argument to see if it is less than the second.

Format

[!UGT integer1 integer2]

...

...

[[!ELSE]]

...

...

[!END]

This pseudomacro compares two arguments, each of which must be an unsigned decimal integer value in the range 0 through 4,294,967,295.

If integer1 is less than integer2, the CLI executes the statements that follow !UGE until the corresponding !ELSE or !END.

Otherwise, the CLI ignores the statements up to the corresponding !ELSE or !END. If the sequence includes an !ELSE statement, the CLI executes the statements that follow !ELSE up to the corresponding !END.

- No templates.
- No argument switches.
- No macro name switches.
- Requirement: *Standard.*
- See also: !ELSE, !END, !UEQ, !UGE, !UGT, !ULE, !UMAXIMUM, !UMINIMUM, !UNE, VAR and !VAR (CLI32 only), VARn and !VARn.

Why Use It?

Use the !ULT pseudomacro to compare two integer values, and use this information to determine whether or not to perform a sequence of statements.

!ULT Example

The macro CENT.CLI converts a Fahrenheit temperature to a roughly equivalent Centigrade temperature. The macro is based on the formula

C = 5/9 x (F–32)

The macro uses the !ULT pseudomacro to display a message if the input temperature is less than 32 degrees F.

```
COMMENT This is macro CENT.CLI.
[!ult,%1%,32]
        write Unfortunately this macro cannot handle temperatures under 32 F.
[!else]
        write %1% Fahrenheit is &
        [!udivide,[!umultiply,5,[!usubtract,%1%,32]],9] Centigrade.
[!end]
```

Sample dialog with the macro follows.

```
) CENT 40 ↵
40 Fahrenheit is 4 Centigrade.

) CENT 80 ↵
80 Fahrenheit is 26 Centigrade.

) CENT 81 ↵
81 Fahrenheit is 27 Centigrade.

) CENT 82 ↵
82 Fahrenheit is 27 Centigrade.
```

!UMAXIMUM *Pseudomacro*

Expands to the maximum value of the arguments (CLI32 only).

Format

[!UMAXIMUM *[integer] [...]*]

This pseudomacro expands to the largest value of the specified integers. Each integer must be unsigned decimal value between 0 and 4,294,967,295. If there are no integer arguments, the pseudomacro returns 0. If there is exactly one argument, the pseudomacro returns the value of the argument.

Why Use It?

Use this pseudomacro to find the largest number in a set of unsigned integers. You may need this number — particularly with VARn variables — to perform various arithmetic operations.

- No macro name switches.
- No argument switches.
- Requirement: *Standard.*
- See also: !ELSE, !END, !UEQ, !UGE, !UGT, !ULE, !UMINIMUM, !UNE VAR and !VAR, VARn and !VARn.

!UMAXIMUM Examples

Next are several WRITE statements that show the behavior of pseudomacro !UMAXIMUM.

```
) WRITE [!UMAXIMUM 68 14 2 643] ↵
643
) VAR0 988 ↵
) WRITE [!UMAXIMUM 69 8 3 0 8 5 195 [!VAR0] 6 3 0] ↵
988
) WRITE [!UMAXIMUM 194] ↵
194
) WRITE [!UMAXIMUM] ↵
0
```

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 maximum is 8, which is what the CLI displays.

```
comment Macro COMPARE.CLI — compares two files and displays
comment length of the larger.
var0 [!size %1%]
var1 [!size %2%]
write The larger file holds [!umaximum [!var0],[!var1]] bytes.
```

This macro compares the sizes of two files.

!UMINIMUM *Pseudomacro*

Expands to the minimum value of the arguments (CLI32 only).

Format

[!UMINIMUM *[integer] [...]*]

This pseudomacro expands to the smallest value of all specified integers. Each integer must be an unsigned decimal value between 0 and 4,294,967,295. If there are no arguments, the pseudomacro returns 4,294,967,295. If there is exactly one argument, the pseudomacro returns the value of the argument.

Why Use It?

Use this pseudomacro to find the smallest number in a set of unsigned integers. You may need this number to perform various arithmetic operations.

- No macro name switches.
- No argument switches.
- Requirement: *Standard.*
- See also: !ELSE, !END, !UEQ, !UGE, !UGT, !ULE, !UMAXIMUM, !UNE VAR and !VAR, VARn and !VARn

!UMINIMUM Examples

Next are several WRITE statements that show the behavior of pseudomacro !UMINIMUM.

```
) WRITE [!UMINIMUM 7 8 94 0 43] ↵
0

) WRITE [!UMINIMUM 68 14 2 643] ↵
2

) WRITE [!UMINIMUM 8 3 0 8 5 6 3 0] ↵
0

) WRITE [!UMINIMUM 194] ↵
194

) WRITE [!UMINIMUM] ↵
4294967295

) VAR0 988 ↵
) WRITE [!UMINIMUM 69 8 3 0 8 5 195 [!VAR0] 6 3 0] ↵
0
```

```
) 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.

!UMULTIPLY Example 2

```
) WRITE [!UMULTIPLY 256 12 100] ↵
307200
```

With CLI32, this command displays the result of multiplying the value 256 by the value 12, and then multiplying that value by 10.

!UMULTIPLY Example 3

The macro SQUARE.CLI displays the result of multiplying a numeric argument by itself. The !UMULTIPLY pseudomacro can display a result as large as 4,294,967,295; SQUARE.CLI accepts arguments less than or equal to the square root of this value.

```
comment This is macro SQUARE.CLI.
[!ule %1% 65535]
        write %1% squared is [!umultiply %1% %1%].
[!else]
        write This macro can't square a number greater than 65535.
[!end]
```

UNBLOCK *Command*

Unblocks a previously blocked process.

Format

UNBLOCK { process–ID | processname | username:processname } *[...]*

This command unblocks a process that was previously blocked by the BLOCK command.

You must supply a process ID or a process name. You can use a process name only if the process was created with the PROCESS command and /NAME=name switch.

- No templates.
- No argument switches.
- Requirement: *Standard* to unblock a subordinate process; *PID 2* or *Superprocess* to unblock any blocked process.
- See also: BLOCK, PROCESS, SUPERPROCESS, PRIVILEGE SUPERPROCESS (CLI32 only).

Why Use It?

Use the UNBLOCK command to release a blocked process: a process you (or, if you have Superprocess or are PID 2) has previously blocked with the BLOCK command.

The PED utility can tell you which processes are blocked. PED is described in *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 switch /STR=.

UNBLOCK Example 1

```
) PROCESS/NAME=PROGA  :UDD:SMITH:PROGA ↵
PID 17

) BLOCK 17 ↵
...
...
) UNBLOCK 17 ↵
```

The first command creates a new process, to which the system assigns PID 17. The next command blocks that process, perhaps to perform related operations. Later, the UNBLOCK command releases the blocked process.

In this example, the following command would be equivalent to the UNBLOCK 17 command.

```
) UNBLOCK  SMITH:PROGA ↵
```

UNBLOCK Example 2

```
) SUPERPROCESS  ON ↵
+) UNBLOCK  DATA_SORT  ↵
+) SUPERPROCESS  OFF ↵
)
```

The UNBLOCK command releases the blocked process named DATA_SORT. Because the process was not a subordinate process, Superprocessmode is needed.

UNBLOCK Example 3

```
) PROCESS/NAME=JD JMP_DOT.PR ↵  (Creates a process with name JD to run
PID: 31                            program JMP_DOT.PR.)

) WHO  JD ↵                     (Displays information about JD.)
PID: 31  KAY  JD  ...  ..

) RUNTIME  JD ↵                 (Gets runtime information about JD.)

) BLOCK  JD ↵                   (Blocks process JD.)

) UNBLOCK  JD ↵                 (Unblocks process JD.)
```

!UNE *Pseudomacro*

Tests two unsigned decimal integer arguments for inequality.

Format

```
[!UNE integer1  integer2]
        ...
        ...
[ [!ELSE] ]
        ...
        ...
[!END]
```

This pseudomacro compares two arguments, each of which must be an unsigned decimal integer value in the range 0 through 4,294,967,295.

If the arguments are unequal, the CLI executes the statements that follow !UNE until the corresponding !ELSE or !END.

Otherwise, the CLI ignores the statements up to the corresponding !ELSE or !END. If the sequence includes an !ELSE statement, the CLI executes the statements that follow !ELSE up to the corresponding !END.

- No templates.
- No argument switches.
- No macro name switches.
- Requirement: *Standard*.
- See also: !ELSE, !END, !UEQ, !UGE, !UGT, !ULE, !UMAXIMUM, !UMINIMUM, !UNE, VAR and !VAR (CLI32 only), VARn and !VARn.

Why Use It?

Use the !UNE pseudomacro to test two integer values for inequality, and use this information to determine whether or not a sequence of statements is to be performed. If you want to compare string values, use !NEQUAL instead.

!UNE Example

(within a macro)

```
[!une, [!size %1%], [!size %2%] ]
        write The files are unequal in length.
[!end]
```

The first statement compares the sizes of two files passed as arguments to the macro. If the file sizes are unequal, the macro displays the message *The files are unequal in length*.

UNLOCK *Command*

Frees a locked CLI.

Format

UNLOCK [*CLI–command–to–enable*[1] | /CX *[EXEC–command–to–enable]*[1]]

[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).

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

[!USUBTRACT { integer1 integer2 / integer integer [...][1] }]

[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).

VAR (continued)

/LEVEL=n

Sets a named variable to the value it has in the specified environment level, n. With the /ALL switch, sets *all* named variables to the values they had at the specified environment level.

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.

/NAME=variable–name

Names the variable. The name must have between 1 and 32 characters, each of which is a legal filename character. This switch is required.

/P*[=n]* or /PREVIOUS*[=n]*

Without =*n*, expands to the value a string has at the previous environment level. With =*n*, sets the named variable to the value it has at absolute level n. With the /ALL switch, sets all named variables to the values they had at the specified environment level.

VAR Examples

Sample dialog with the VAR command follows. Text to the right in parentheses is commentary.

```
) VAR/NAME=ONE 123 ↵          (At Level 0.)
) VAR/NAME=ONE ↵
123

) VAR/NAME=FOUR 57 ↵

) PUSH ↵                      (To Level 1.)
) VAR/NAME=ONE 456 ↵          (Assign ONE a value here.)
) VAR/NAME=ONE/PREVIOUS ↵     (Assign ONE its value at Level 0.)
) VAR/NAME=ONE ↵              (Display ONE's value.)
123

) VAR/NAME=ONE 456 ↵          (Assign ONE a new value)
) VAR/NAME=FOUR 899 ↵         (... and FOUR)
) VAR/NAME=HELLO 77889 ↵      (... and HELLO)
```

VAR (continued)

```
) PUSH ↵                          (To Level 2.)
) VAR/NAME=ONE 789 ↵              (Assign ONE a value here.)
) VAR/NAME=ONE/LEVEL=0 ↵          (Assign ONE its value at Level 0.)
) VAR/NAME=ONE ↵                  (Display ONE's value.)
123

) VAR/NAME=ONE/LEVEL=-1 ↵         (Assign ONE its value at Level 2-1.)
) VAR/NAME=ONE ↵                  (Display ONE's value.)
456

) VAR/NAME=ONE 789 ↵              (Assign ONE a value here at Level 2.)
) VAR/NAME=HELLO 4 ↵              (Assign HELLO a value here.)
) VAR/INFO ↵                      (Display all variables' values here.
FOUR = 899                        (FOUR's value is as passed from Level 1).
HELLO = 4
ONE = 789

) VAR/INFO/NAME=ONE ↵             (Display ONE's values at all levels.)
ONE (0) = 123                     (ONE's value at Level 0.)
ONE (1) = 456                     (ONE's value at Level 1.)
ONE (2) = 789                     (ONE's value at Level 2.)

) VAR/INFO/ALL ↵                  (Display all variables' values
                                  at all levels, in alphabetical
                                  order by variable name.)
FOUR (0) = 57                     (FOUR's value at Level 0.)
FOUR (1) = 899                    (FOUR's value at Level 1.)
HELLO (1) = 77889                 (HELLO's value at Level 1.)
HELLO (2) = 4                     (HELLO's value at Level 2.)
ONE (0) = 123                     (ONE's value at Level 0.)
ONE (1) = 456                     (ONE's value at Level 1.)
ONE (2) = 789                     (ONE's value at Level 2.)

) VAR/NAME=YYY ↵                  (Variable YYY has not received a value.)
0                                 (The CLI assigns it 0.)
```

!VAR *Pseudomacro*

Expands to the value of the specified integer variable (CLI32 only — !VARn for both CLIs follows the command VARn).

Format

[!VAR/NAME=variable–name]

This CLI32 pseudomacro gives you access to the values of the variables you previously defined with commands of the form VAR/NAME=variable–name value.

The value that [!VAR] expands to is an integer between 0 and 4,294,967,295. The variable name can be any legal AOS/VS or AOS/VS II filename.

The !VAR pseudomacro exists in addition to — not as a replacement of — !VARn pseudomacros. The !VAR/NAME= variables and the !VARn variables simply differ. For example, the two commands

```
) VAR/NAME=VAR0  83 ↵
) VAR0  38 ↵
```

assign values to two different variables that happen to have the same name (VAR0) in different contexts — the VAR command and the VARn command.

There is no conflict between the !VAR pseudomacro of CLI32 and the !VARn pseudomacro. While it is possible to have common names, they expand to different values. For example,

```
) WRITE  [!VAR/NAME=VAR0] ↵
) WRITE  [!VAR0] ↵
```

The two commands display values stored in two different places.

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.
- No argument switches.
- Accepts macro name switches (described later)
- Requirement: *Standard.*
- See also: VAR, VARn.

Why Use It?

Use the !VAR pseudomacro to obtain the values of named variables in macros with CLI32. (With CLI16, you must use unnamed variables and the !VARn pseudomacro.)

!VAR Macro Name Switches

/LEVEL=n	Expands to the value a named variable has in the specified environment level. With the /ALL switch, sets *all* unmamed variables to the values they had at the specified level. 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.
/NAME=variable–name	You must supply this switch to specify the the variable name. Variable (and string) names havee between 1 and 32 filename characters.
/P[=n] or /PREVIOUS*[=n]*	Without =*n*, expands to the value the named variable has at the previous environment level. With =*n*, expands to the value the named variable has at the specified environment level. The *n* specifies the number of levels above the current level (toward 0). With the /ALL switch, sets all named variables to the values they had at the givwn environment level.

!VAR Examples

Text to the right of the sample dialog with the !VAR pseudomacro is commentary.

```
) VAR/NAME=XYZ 123 ↵                          (At Level 0.)
) WRITE [!VAR/NAME=XYZ] ↵
123
) PUSH ↵                                      (To Level 1.)
) WRITE [!VAR/NAME=XYZ] ↵
123
) VAR/NAME=XYZ [!VAR/NAME=XYZ]456 ↵           (Assign the concatenation of
) VAR/NAME=XYZ ↵                              [!VAR/NAME=XYZ], which is 123, and
123456                                        456 — the result is 123456.)
) WRITE [!VAR/NAME=XYZ/PREVIOUS] ↵            (Display the value of XYZ at the
123                                           previous level — 0.)
) PUSH ↵                                      (To Level 2.)
) VAR/NAME=XYZ 456 ↵                          (Assign XYZ a new value at Level 2.)
) WRITE [!VAR/NAME=XYZ] ↵                     (XYZ's value at Level 2.)
456
) WRITE [!VAR/NAME=XYZ/LEVEL=0] ↵             (Display value of XYZ at Level 0.)
123
) WRITE [!VAR/NAME=XYZ/LEVEL=–1] ↵            (Display value of XYZ at Level 2
123456                                        minus 1 — Level 1.)
) WRITE [!VAR/NAME=YYY] ↵                     (Display value of YYY.)
0                                             (Variable YYY has not received a value,
                                              so the CLI assigns it 0.)
```

VARn *Command*

Displays or sets the value of CLI variable VARn.

Format

VARn *[integer-or-expression-with-pseudomacros]*

CLI16 and CLI32 include 10 variables, VAR0 through VAR9. Each variable can represent an integer value ranging from 0 through 4,294,967,295.

Each CLI environment level has its own set of variables. So, the value of a variable can differ from one level to another. When you move down a level (with the PUSH command) the initial value for that level's variables is the value assigned in the previous environment (the one you just left). In Level 0, the initial value for these variables is 0. (See Example 4.)

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

Why Use It?

Use the VARn command to assign a value or display the value of a CLI variable. To insert the value of a variable within a macro, use the !VARn psuedomacro.

(In CLI32, to help identify variables, you might choose to use named variables — VAR and !VAR — described earlier, instead of VARn and !VARn.)

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.

/LEVEL=n (CLI32 only.) Replaces the current value ofthe variable with the value it has in 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.

VARn (continued)

/P*[=n]*	Without =*n*, replaces the current value of the variable with the value it has in the previous CLI environment. (Omit the integer argument.) With =*n* (CLI32 only), expands to the value the string has at the specified environment level. The *n* specifies the number of levels above the current level (toward 0).
/PREVIOUS*[=n]*	(CLI32 only.) Same as /P.

VARn Example 1

```
) VAR2 ↵
0
```

This command displays the current value of VAR2.

VARn Example 2

```
) VAR0 17 ↵
```

This command assigns the value 17 to VAR0.

VARn Example 3

```
) VAR2  34573 ↵
) WRITE [!UADD [!VAR2] 14] ↵
34587
```

The first command assigns the value 34573 to VAR2. The next command displays the result of adding the value 14 (via the !UADD pseudomacro) to the pseudomacro !VAR2, which substitutes the current value of the variable VAR2.

VARn Example 4

```
) VAR8 64 ↵          (Sets VAR8 to the value 64.)
) IEVEL ↵  (Displays the current CLI environment level.)
Level 0
) PUSH/V ↵ (Moves down to the next level.)
Level 1
) VAR8 ↵   (Displays the value of VAR8 in Level 1.)
64         (The initial value is that of the previous environment.)
) VAR8  72 ↵         (Sets VAR8 to the value 72.)
) POP/V ↵  (Moves up to the previous environment.)
Level 0
) VAR8 ↵   (Displays the value of VAR8.)
64
```

!VARn *Pseudomacro*

Expands to the current value of VARn.

Format

[!VARn]

The 10 pseudomacros !VAR0 through !VAR9 represent the current value of the CLI variables VAR0 through VAR9.

Each CLI environment level has its own set of variables. So, the value of a variable can differ from one level to another. When you move down a level (with the PUSH command) the initial value for that level's variables is the value assigned in the previous environment (the one you just left). The initial value for these variables is 0.

- No arguments.
- Accepts macro name switches (described later).
- Requirement: *Standard.*
- See also: VARn.

Why Use It?

Use the !VARn pseudomacro to include the current value of a CLI variable in a command.

(In CLI32, to help identify variables, you might choose to named variables — VAR and !VAR — described earlier, instead of VARn and !VARn.)

Macro Name Switches

/LEVEL=n — (CLI32 only.) Expands to the value the the variable has in 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*, expands to the value the variable has in the previous CLI environment level. With =*n* (CLI32 only), expands to the value the variable has at the specified environment level. The *n* specifies the number of levels above the current level (toward 0).

/PREVIOUS*[=n]* — (CLI32 only.) Same as /P.

!VARn Example 1

```
) VAR2  39 ↵
) WRITE The current value of VAR2 is [!VAR2] ↵
The current value of VAR2 is 39
```

This command displays a statement that reports the value of VAR2.

!VARn Example 2

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 statement prompts for the dividend value, assigning it to the CLI variable VAR0. The second statement requests the divisor value, assigning it to CLI variable VAR1. The third statement uses the !VAR0 and !VAR1 pseudomacros 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.

WHO *Command*

Displays process information.

Format

WHO [*[hostname:]process-ID* | *processname* | *username:processname* | *!PIDS*] *[...]*

This command displays the process ID (PID), username, process name, and program name of the specified process.

You can supply a process ID, a process name, or the [!PIDS] pseudomacro. You can use either process name form, but only if the process was created with the PROCESS command and /NAME=name switch.

If your system is on a XODIAC/XTS network, you can specify a hostname to obtain information about processes on other systems.

If you omit the argument(s), the command applies to the current CLI.

- No templates.
- No argument switches.
- Requirement: *Standard.*
- See also: !PID, !USERNAME, WHOS.

Why Use It?

Use the WHO command to obtain information about the process ID (PID) or username of a process. For example, you can use it to learn which CLI you are running (CLI16 or CLI32) or the PID assigned to your CLI. If you want to know who is using an unattended terminal that is running the CLI, you can type this command at that terminal.

If you receive a message from another user, but the message shows only the sender's PID number, not a username, you can use the WHO command with that PID number to discover who sent you the message.

To list all processes on the system, use the WHOS macro, next.

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=.

WHO Example 1

```
) WHO ⤶
PID:  17  TEDDY    CON17    :CLI16.PR
```

This command displays information about the current CLI process: it is PID 17, the username is TEDDY, the terminal filename is CON17, and the program is CLI16.

WHO Example 2

The following message appears on your screen:

From Pid 12: Please come to my office when you get a chance.

To discover who sent the message, you would enter the following command and see a similar response.

```
) WHO 12 ⤶
PID:  12    LINDSAY    CON67    :CLI.PR
```

WHO Example 3

```
) WHO  CENTRAL:[!PIDS] ⤶
```

This command, issued on a system that is part of an XTS network, checks the PIDs on a system whose hostname is CENTRAL.

WHOS.CLI *Macro*

Displays information on all processes on your system.

Format

WHOS *[hostname]*

This macro, supplied with the operating system, displays the process ID (PID), username, process name, and program name of all processes on your system.

If your system is on a XODIAC/XTS network, you can specify a hostname to obtain runtime information for that system; the CLI will display runtime information for that system followed by the processes on your system.

- No templates.
- Requirement: *Standard* (you must have Execute access to directory :UTIL and Read access to WHOS.CLI — these are true by default).
- See also: !PID, !USERNAME, WHO.

Why Use It?

Use the WHOS macro command to obtain information about all processes on the system — perhaps so you can send someone a message.

Macro 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 and with this macro. That section also explains the CLI32 switch /STR=.

WHOS Example

```
) WHOS ↵
Elapsed 130:09:02, CPU  0:05:47.738, I/O Blocks 1682, Page Secs 181437
PID:    1 PMGR     PMGR        :PMGR.PR
PID:    2 OP       OP          :CLI32.PR
PID:    3 OP       EXEC        :UTIL:EXEC.PR
PID:    4 OP       XBAT1       UTIL:XLPT.PR
PID:    5 OP       XMNT1       :NET:XMNT.PR
PID:    6 OP       XLPT1       :UTIL:XLPT.PR
PID:    7 OP       NETOP       :NET:NETOP.PR
.
PID:    40 TOMR    CON159      :CLI32.PR
PID:    43 LEVITT  WP_5110     UTIL:CEO_DIR:CEO_WP.PR
PID:    56 LEVITT  CON69       :CLI32.PR
PID:    57 LEVITT  CEO_CP_56   :UTIL:CEO_DIR:CEO_CP.PR
.
```

This macro displays information all processes on the system.

WRITE *Command*

Displays arguments or writes them to a file.

Format

WRITE { *[argument]* | /FILEID=file–ID *[argument]*[1] }

1 CLI32 only.

The first form of the WRITE command displays the specified arguments on @OUTPUT (normally the terminal screen), unless you direct the display to a list file.

The second form of the command writes specified arguments to the file you specify with the /FILEID= switch. This file must have been opened for writing earlier (via the form OPEN/WRITE pathname); the OPEN command without an argument shows the file IDs of all open files. By default, the file–ID is the filename without trailing suffix.

The arguments can be text strings, pseudomacros, or CLI command lines.

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

Why Use It?

Use the WRITE command if you want to write arguments to the screen or to a list file. This command is useful in macros to give status information. You can also use the WRITE command to verify a template expansion or to preview the results of a command; see Examples 3 and 4. This previewing of complete commands with angle brackets or parentheses is one of the most valuable uses of the WRITE command.

With CLI32, you can use WRITE/FILEID= to write sequential lines of text to a file; and you can also use it to create files with special control characters (as explained in the /NONEWLINE switch).

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.

WRITE Example 1

```
) WRITE [!OCTAL 96] ↵
140
```

This command writes the octal equivalent of 96 (decimal) to the screen.

WRITE Example 2

(within a macro)

```
write Your username is [!username] and your PID is [!pid].
```

This command causes the macro to display the username and PID of the process that executed the macro. The output will look something like this:

Your username is MARY and your PID is 315.

WRITE Example 3

```
) WRITE (,OLD)FILE.<PR ST> ↵
FILE.PR  FILE.ST
OLDFILE.PR  OLDFILE.ST
```

This WRITE command shows how the CLI expands an argument that includes parentheses and angle brackets. Thus, you can use the WRITE command to verify that the CLI will expand an argument the way you want it to.

WRITE Example 4

```
) WRITE XEQ SCOM <,:ARCHIVES:>MYPROG.F77 ↵
XEQ SCOM MYPROG.F77 :ARCHIVES:MYPROG.F77
```

This WRITE command shows how the CLI expands a command with an argument that includes angle brackets. Thus, you can use the WRITE command to preview the results of a command before you actually give the command to the CLI. If you are satisfied with these results, you can obtain them simply by giving the following keystrokes.

CTRL–A	(Redisplay the WRITE command.)
CTRL–H	(Move the cursor to the W of WRITE.)
	(Press the space bar five times to delete WRITE.)
↵	(Give the command to the CLI.)

WRITE Example 5 (CLI32 only)

The following commands place three lines into a freshly created file, close it, reopen it, and then display the file's records.

```
) OPEN/DELETE/WRITE  MYFILE ↵
MYFILE
```

The OPEN command deletes file MYFILE if it already exists and then creates and opens it for writing. The CLI displays the file–ID, which (here) is the same as the filename. Future READ, WRITE, and CLOSE commands can use the file–ID MYFILE to identify the file.

```
) WRITE/FILEID=MYFILE/FORCE  The working directory is [!directory].↵
) WRITE/FILEID–MYFILE/FORCE  My PID is [!PID]. ↵
) WRITE/FILEID=MYFILE/FORCE  My username is [!username]. ↵
) CLOSE/ALL ↵
) TYPE  MYFILE ↵
The working directory is :UDD1:JOAN:REPORTS.
My PID is 188
My username is JOAN
```

These commands place three records into file MYFILE and then close the file and show what it contains.

WRITE Example 6

This example consists of four related macros that work together to display a menu of actions, accept a choice, and do the processing itself. Macro USER_MENU.CLI is the driver macro; it displays the menu of possible actions, accepts the response, and invokes one of the macros USER_1.CLI, USER_2.CLI, or USER_3.CLI to do the actual processing.

The four macros use the WRITE command to position the cursor, display text, and specify different types of display such as bright, dim, underlined, and blinking.

Macro USER_MENU.CLI and the submacros it calls are shown in Figures 5–1 though 5–4.

```
comment  Macro USER_MENU.CLI

[!equal,large,comment]

This driver menu macro allows the user to select from the
following four choices.

 {0}  Bye (return to the CLI)
 {1}  Create or edit a file using the SED text editor
 {2}  Display a file on the screen
 {3}  Display the amount of space in the initial directory

The choices numbered 1, 2, and 3 execute macros USER_1.CLI
USER_2.CLI, and USER_3.CLI; each of these macros calls US-
ER_MENU.CLI when done.  Choice 0 returns to the CLI prompt.

Invoke the macro set by typing

USER_MENU and pressing the NEW LINE key.
[!end]

[!nequal,%0/CONT%,/CONT]
     push
     characteristics/off/st
     prompt pop
[!end]

comment  If this macro was invoked without the /CONT (for con-
tinue)
comment  switch, then execute the PUSH and PROMPT commands.
Also,
comment  set the /ST (simulate tabs) characteristic off.  If it
comment  were on, and this macro tried to position the cursor to
comment  column 9 or row 9 via the !ASCII value 211 in a WRITE
comment  command, the result would probably be incorrect.

class1 WARNING
```

Figure 5–1 WRITE Example, Macro USER_MENU.CLI

```
macro USER_MENU.CLI continued.

comment  Display the menu on a blank screen and get the
comment  user's choice.

write [!ascii 214 210 234 203][!date]
comment   The !ASCII 214 erases the page, !ASCII 210 ensures the
comment   cursor is at the left of the page, !ASCII 234 starts dim
comment   display, and !ASCII 203 enables blinking.

write [!ascii 220 310 200][!time]
comment   The !ascii 220 begins a column and row specification.
comment   310 - 200 = 110 octal = 72 decimal and
comment   200 - 200 =   0 octal =  0 decimal, so the time of day
comment   begins in column 72, row 0.

write [!ascii 220 224 205]&
[!ascii 235 224]&
Menu of Choices&
[!ascii 234 225]
comment   Move the cursor to column 20, row 5 (decimal),
comment   stop the dim display (return to bright), start
comment   underlining, display "MENU OF CHOICES",
comment   start dim display, and stop underlining.

write [!ascii 220 217 207]&
{0},,Bye
comment   Move the cursor to column 15, row 7 (decimal),
comment   and display the first choice.

write [!ascii 220 217 210]&
{1},,Create or edit a file using the SED text editor&
[!ascii 220 217 211]&
{2},,Display a file on the screen&
[!ascii 220 217 212]&
{3},,Display the amount of space in the initial directory
comment   Move the cursor to column 15, row 8 (decimal)
comment   and display the second, third, and fourth choices.

write [!ascii 220 217 215]&
Press any other key to repeat the menu.[!ascii 235 212 212]
comment   Move the cursor to column 15, row 13 (decimal),
comment   display a message, stop displaying in dim mode,
comment   move down two lines.
```

Figure 5–1 WRITE Example, Macro USER_MENU.CLI (continued)

```
Macro USER_MENU.CLI continued.

string [!read ,,,,,,,,,,,,,,,,Enter choice 0 to 3: ]
comment   Get the selection.

[!equal, ([!string]), (0)]
     write [!ascii 220 217 224 216]Goodbye[!ascii 217]
     pause 2
     characteristics/on/st
[!end]
comment   If the selection is 0, display "Goodbye" at the bottom
comment   of the screen in blinking characters for two seconds
comment   and then reset the /ST characteristic.  The macro stops
comment   executing after three more comparisons.

[!equal, ([!string]), (1)]
     USER_1
     comment   The selection is 1, so invoke macro USER_1.CLI to
     comment   emphasize the choice by repeating its text in
     comment   blinking bright mode and executing the SED text
     comment   editor.  When USER_1.CLI finishes, it calls this
     comment   macro, USER_MENU.CLI, with only the /CONT switch.
     %0\%/CONT
[!end]

[!equal, ([!string]), (2)]
     USER_2
     comment   The selection is 2, so call macro USER_2.CLI to
     comment   emphasize the choice by repeating its text in
     comment   blinking bright mode and getting the name of the
     comment   file to display. When USER_2.CLI finishes, it calls
     comment   this macro, USER_MENU.CLI with only the /CONT switch.
     %0\%/CONT
[!END]

[!equal, ([!string]), (3)]
     USER_3
     comment   The selection is 3, so invoke macro USER_3.CLI to
     comment   emphasize the choice by repeating its text in
     comment   blinking bright mode and displaying the amount
     comment   of disk space used and available in the initial
     comment   directory.  It should be a control point
     comment   directory.  When USER_3.CLI finishes, reinvoke this
     comment   macro -- USER_MENU.CLI -- with only the /CONT switch.
     %0\%/CONT
[!end]
```

Figure 5–1 WRITE Example, Macro USER_MENU.CLI (continued)

```
Macro USER_MENU.CLI continues.

[!nequal, ([!string]), (0)]
     %0\%/CONT
[!end]
comment    If the String contains 1, 2, or 3, then one of the
comment    preceding three !EQUAL ... !END blocks calls
comment    USER_MENU.CLI.
comment
comment    If the String contains 0, then neither the three
comment    preceding !EQUAL ... !END blocks nor the preceding
comment    !NEQUAL ... !END block calls USER_MENU.CLI and control
comment    returns to the CLI.
comment
comment    If the String contains anything but 1, 2, 3, or
comment    0, the preceding !NEQUAL ... !END block calls
comment    USER_MENU.CLI with only the /CONT switch.

comment    ***** End of Macro USER_MENU.CLI *****
```

Figure 5–1 WRITE Example, Macro USER_MENU.CLI (concluded)

```
comment      Macro  USER_1.CLI

[!equal,large,comment]
This macro is called from the main menu macro, USER_MENU.CLI.
See USER_MENU.CLI for a summary of what macro USER_1.CLI does.

[!end]

push

write [!ascii 220 217 210 216]&
{1},,Create or edit a file using the SED text editor&
[!ascii 217 220 210 226]

pause 2
xeq sed [!read Enter name of the file you want to create or
edit:,,]
pop

comment ***** End of Macro USER_1.CLI *****
```

Figure 5–2 WRITE Example, Macro USER_1.CLI

```
comment                Macro USER_2.CLI

[!equal,large,comment]
This macro is called from the main menu macro, USER_MENU.CLI.
See USER_MENU.CLI for a summary of what macro USER_2.CLI does.
[!end]

push

write [!ascii 220 217 211 216]&
{2},,Display a file on the screen&
[!ascii 217 220 210 226]

pause 2

comment   Temporarily restore the simulation of tabs that
comment   USER_MENU.CLI turned off.  This will display any tabs
in
comment   the file properly.

characteristics/on/st

type   [!read Enter the name of the file you want to display:,,]

comment   Reset the simulation of tabs.

characteristics/off/st

comment ***** End of Macro USER_2.CLI *****
```

Figure 5–3 WRITE Example, Macro USER_2.CLI

```
comment      Macro  USER_3.CLI

[!equal,large,comment]

This macro is called from the main menu macro, USER_MENU.CLI.
See USER_MENU.CLI for a summary of what macro USER_3.CLI does.
[!end]

push

write [!ascii 220 217 212 216]&
{3},,Display the amount of space in the initial directory&
[!ascii 217 220 210 226]

pause  2
space :udd:[!username]
pause  2
pop

comment ***** End of Macro USER_3.CLI *****
```

Figure 5–4 WRITE Example, Macro USER_3.CLI

XEQ *Command*

Executes a program.

Format

XEQ pathname*[.PR] [argument] [...]*

The command executes the program (specified by pathname) as a son (subordinate) swappable process of the CLI. The CLI first looks for pathname.PR; if that does not exist, the CLI looks for pathname. The pathname must be an executable program.

If the program requires or accepts arguments, you can specify them. For information about how a program started from the CLI can gain access to the arguments and switches included in the CLI command line, see the ?GTMES system call in the *AOS/VS, AOS/VS II, and AOS/RT32 System Call Dictionary, ?A through ?N.*

The new process has the same priority and privileges as its parent CLI process. It also uses the same generic files, with the possible exception of @LIST and @DATA; the new program uses the current list file and data file settings, which may be different from the CLI's.

The CLI process blocks until the program terminates. If the program returns an error condition, the CLI tries to interpret the error code.

XEQ works the same way as EXECUTE.

Program Input

The program's use of @INPUT depends on whether you execute the program in batch mode or not. In batch mode, you can use the /I or /M switch to provide input. If you are working at a terminal, you can enter input through the terminal.

- No templates.
- Use any argument switches appropriate for the program.
- Requirement: *Standard* (Execute access to the program file and to its parent directory or directories).
- See also: PROCESS (to specify detailed process information) and CHAIN (to overwrite the CLI with a new process).

Why Use It?

Use the XEQ (or EXECUTE) command to run a program, and then return to the CLI when the program terminates.

NOTE: You do not use this command to run a macro. To execute a macro, simply type the macro name.

XEQ (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 the macro file first.

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=.

/I	Creates input for the program from @INPUT. If you use pseudomacro input, the CLI will ignore rather than interpret. 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 (any pseudomacros within the macro are ignored, not interpreted). 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.

XEQ Example 1

```
) XEQ  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.

XEQ Example 2

```
) XEQ  MYPROG  0  1 ↵
```

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

XEQ Example 3

```
) XEQ/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. When the program ends, you can display the string to learn the program termination message.

End of Chapter

Chapter 6
Using Magnetic Tape and Labeled Diskettes

This chapter explains how you can use magnetic tape and diskettes to store or retrieve information. This chapter is intended for users who want to keep their files on one of these storage media.

If you want to do system backups, read about backups in *Managing AOS/VS and AOS/VS II.*

The major sections in this chapter proceed as follows.

- About Magnetic Tape
- Using Unlabeled Magnetic Tape
- Using Labeled Magnetic Tape
- Acting as System Operator
- Summary of Tape Operations
- About Diskettes

About Magnetic Tape

Magnetic tape can be on a reel or in a cartridge. If you are unsure about the type of magnetic tape you can use at your site, see your system manager.

Tapes, unlike disks, allow only sequential read/write access. To locate a specific file on tape, for example, the system must begin the search at the beginning of the tape file. Similarly you cannot *insert* data on a tape; if you write information to tape, all data following that position is effectively lost.

A magnetic tape can be either labeled or unlabeled.

An *unlabeled tape* can contain as many as 100 tape files (numbered 0 through 99) on a single volume; file numbers start over at 0 on each physical tape. A tape file can contain one or more actual files. (You can, for example, dump your entire directory structure to a single tape file.) To write to or read from unlabeled tape, you must specify the number of the tape file you want to use.

If you use the DUMP or LOAD command with unlabeled tape, a dump or load operation is restricted to the capacity of one physical tape (reel or cartridge). If you use the DUMP_II or LOAD_II utility, a dump or load operation can span multiple tapes; the program will prompt you to mount new tapes as it needs them. However, DUMP_II and LOAD_II do not keep track of sequence numbers, so you must be very careful to place accurate paper labels on the tapes, so they can be loaded in the correct order. For the most precise multiple–volume dumps and loads, use labeled tapes.

A *labeled tape* begins with a block of information that identifies the contents of the tape. Among other things, the label specifies the filename for the tape set, an expiration date for the tape set, and volume ID (often called *volid*), which uniquely identifies the tape reel or cartridge. You can specify multiple volume IDs when you request that the tape set be mounted (in the MOUNT command). Thus, a labeled tape dump file can span multiple tape volumes.

To dump to or load from a labeled tape, you must specify its volume ID. This not only helps you catalog and protect your data, but also ensures that the correct tape is being used. Labeled tapes do not use tape file numbers; to access the information on the tape, you use a filename. This filename can be more descriptive than the tape file number you must use with unlabeled tape.

Using Unlabeled Magnetic Tape

How you use unlabeled tape depends on whether you will mount the tape yourself or request the system operator to do it for you. The following sections describe each case.

Using Unlabeled Tapes without a System Operator

If you mount your own tapes without the intervention of the system operator, you must note the type and unit number of the tape unit on which you mount the tape.

To refer to a tape unit, specify its device name followed by the unit number. In AOS/VS, device names are rigidly defined; in AOS/VS II, the system manager can select any names he or she wants during system generation. The default tape unit device names, based on type and and capacity, are as follows.

Unit Type	Default Device Name
Dual density, 1600 or 800 bytes per inch, ECLIPSE bus	@MTB
Single density, 1600 bytes per inch, reel or cartridge	@MTC
Dual density, 6250 or 1600 bytes per inch	@MTD
Cartridge, quarter or half inch, 6400 bytes per inch	@MTJ
Dual density, 1600 or 6250 bytes per inch, on MRC bus	@MRCTAPE

To determine which tape units are available on your system, type

```
) FILESTATUS/TYPE=MTU @MT+ ↵
```

The output will list the tape units defined on your system.

Files on unlabeled tapes are numbered from 0 through 99. To refer to a specific tape file, append the file number, preceded by a colon, to the tape unitnumber. So, to refer to the first file on the tape mounted on unit MTD0, you would specify @MTD0:0; the second file on that tape is MTD0:1, and so on.

For example, to dump all files in your working directory to the first file of the tape mounted on unit MTB2, you would type a command like

```
) DUMP_II/V @MTB2:0 ↵
```

Then, if you wanted to load all the program files contained in the second tape file of the tape mounted on unit MTB1, you would type a command like

```
) LOAD_II/V @MTB1:1 +.PR ↵
```

DUMP_II and LOAD_II let you use multiple tapes. The program will prompt you for a new tape if one is needed. This advantage should prompt you to use DUMP_II/LOAD_II (instead of DUMP/LOAD) whenever possible. However, DUMP_II and LOAD_II do not keep track of sequence numbers, so you must be very careful to place accurate paper labels on the tapes, so they can be loaded in the correct order. (Tape file numbers start over at 0 when I/O continues on the next tape volume). For the most precise multiple–volume dumps and loads, use labeled tapes (with DUMP_II or DUMP, or LOAD_II or LOAD).

Using Unlabeled Tapes with a System Operator

At some sites, users do not have physical access to tape units; only system operators can mount and dismount tapes. In this case you must use the MOUNT command to request the operator to mount the tape(s) you want to use on a tape unit. (The MOUNT command is described fully in Chapter 5.)

The MOUNT command, among other things, sends a message requesting a tape mount to the system console. But unless someone is acting as an operator at the system console, the MOUNT request will wait indefinitely. You can check for the presence of the system operator by typing

```
) WRITE [!OPERATOR] ↵
```

The CLI reports whether the operator is on or off duty. (The EXEC command OPERATOR ON and OPERATOR OFF sets the value to ON or OFF — so an ON value does not guarantee that an operator is on duty.)

When you issue a MOUNT request, you specify a *linkname,* which you will subsequently use to refer to the mounted tape. The name you use cannot refer to an existing file. After mounting the tape, the operator tells the system which unit is being used. The system then creates your link file and associates it with that tape unit. Your linkname will then automatically refer to the appropriate tape unit.

The system creates the link file in your initial directory (unless your MOUNT command uses the /DIRECTORY switch to specify another directory). When you subsequently use the link filename in a DUMP/DUMP_II or LOAD/LOAD_II command, be sure the system can locate the link file. If the link is not in your working directory, you must make sure the system can find it: either supply a full pathname to the link file, or place the directory from which you type the MOUNT command in your search list.

The MOUNT command also lets you include a message to the system operator. You can use this message to ask the operator to insert a write–enable ring, for example.

After you issue the MOUNT request, the operator will either mount the tape or refuse your request. If he or she refuses the request, you will see a *Refused* message. Otherwise, the CLI prompt will return without a message. If you did not use the /NOPEND switch on the MOUNT command, your CLI prompt will not reappear until the operator responds. You can cancel your MOUNT request by typing CTRL–C CTRL–A. You should then delete the link file from your initial directory.

After the operator mounts the tape you want, he or she confirms the mount by typing a command to EXEC at the system console. You can then issue commands (like DUMP_II or LOAD_II) to write to or read from the tape(s).

When you have finished with the tape, you must use the DISMOUNT command to ask the operator to dismount the tape. DISMOUNT, like MOUNT, is fully described in Chapter 5.

Examples with Unlabeled Tape and the MOUNT Command

To request the operator to mount an unlabeled tape, you could type the command

```
) MOUNT NEWTAPE Please mount a scratch tape with ring in ↵
```

This posts the request to the system operator at the system console. Since you omitted the /NOPEND switch, your CLI waits for someone at the system console to respond. Assume the operator mounts the tape.

To use the tape, specify the linkname (in this case NEWTAPE), and append the tape file number, preceded by a colon. For example,

```
) DUMP_II/V  NEWTAPE:0  +.SR ↵
.
(Program displays filenames dumped)
.
) DUMP_II/V  NEWTAPE:1  +.LS ↵
.
(Program displays filenames dumped)
.
```

The first command dumps to tape file 0 all files ending with .SR in the working directory. The next command dumps files ending in .LS to the second tape file. When you are finished with the tape, you should use the DISMOUNT command to equest the operator to remove the tape from the unit.

```
) DISMOUNT NEWTAPE Thanks – Please return tape to me ↵
```

In the next example, the /NOPEND switch prevents the CLI from waiting until the operator responds to the mount request.

```
) MOUNT/NOPEND  NEWVOL  Please mount scratch tape with ring ↵
Queued, sequence number = 84

) QDISPLAY/TYPE=MOUNT ↵

MOUNTQ  MOUNT   Open
84      TOM
```

(System operator sends message confirming that a tape is mounted.)

```
) DUMP_II/V  NEWVOL:0  +.PR ↵
.
```

(Program displays filenames dumped)

```
.
) DISMOUNT  NEWVOL ↵
```

The QDISPLAY command shows the status of the MOUNT queue, which lists the current mount request as sequence number 84. After dumping files to the tape via the linkname NEWVOL, user TOM asks the operator to dismount the tape. (He could also have used QCANCEL with the sequence number, 84.)

In the next example, the MOUNT command requests the operator to mount a specific tape. After loading program files (.PR) from the first and second tape files, the user asks the operator to dismount and return the tape.

```
) MOUNT  OLDFILES  Please mount backup tape from 24 Nov ↵

 (operator mounts tape)

) LOAD_II/V OLDFILES:0  +.PR ↵
.
```

(displays filenames loaded)

```
.
) LOAD_II/V OLDFILES:1  +.PR ↵
.
```

(displays filenames loaded)

```
.
) DISMOUNT  OLDFILES  Thanks – Please return tape to rack ↵
```

Using Labeled Magnetic Tape

Using labeled tape involves more steps than using unlabeled tape: using them requires someone at the system console, and someone must keep track of the tape labels and volume IDs, tape filenames, and expiration dates. However, labeled tape provides several advantages over unlabeled tape. Labeled tape lets you

- Uniquely identify all volumes in multiple–volume dumps. This helps ensure that the tapes will be loaded correctly if needed in the future. Labeled tapes provide the most secure dumps.
- Describe the contents of a tape set on the tape medium itself. The label on labeled tape has a volume ID and a filename (which is more descriptive than the file number on an unlabeled tape). The label can also include owner and user volume identifying text.
- Protect tape information from being overwritten within a specified period. The system will not write to a tape whose retention period has not expired unless you relabel the tape. By default, DUMP_II and DUMP set a retention period of 90 days from the dump date, but you can a different period with the /RETAIN switch.
- Use the tape with software that requires labels including INFOS II and DG/DBMS.
- Create tapes to be read on non–Data General operating systems such as IBM or any system that uses ANSI–standard labels.

Using labeled tapes requires an operator — someone at the system console — to mount tapes and tell EXEC they are mounted by typing commands at the system console.

Labeling a Tape

The LABEL utility (described in Chapter 5) writes a new label at the beginning of a tape. If a tape has not been labeled, and you want to use it as a labeled tape, you must run LABEL on it. Before labeling the tape, you should ascertain whether or not your site has established conventions for assigning volume IDs and filenames. Following a consistent naming scheme can make it easier to locate information.

IMPORTANT: Writing a new label to a tape that already contains data (whether labeled or not), makes all remaining information on the tape unavailable. The labeling process effectively scratches (blanks) the tape.

To use the LABEL utility you must know

- the name of the tape unit on which the tape to be labeled is mounted
- the volume ID you want to assign to that tape (one to six text characters are allowed).

Switches allow you to specify additional label information, such as IBM format, an owner field, and as many as nine user volume labels.

The form for using the LABEL utility is

XEQ LABEL unitname volid (As always, with CLI32, you can omit XEQ)

For example

) XEQ LABEL @MTB1 VOL001 ↵

For multi–volume labeled tapesets, be sure to make the volume ID of each tape volume unique. And if you plan to dump at a nondefault density, be sure to label the tape using that density. Use the LABEL/DENSITY switch. See the description of the LABEL utility in Chapter 5 for detailed information about switches, and more examples.

After using the LABEL command to label a tape, you should write the label name on an adhesive label and stick it on the tape reel.

Examining Label Contents

If you need to examine the label on a tape volume, use the TYPE command or DISPLAY utility (also described in Chapter 5). Mount the tape with the label you want to check. Then use TYPE or execute DISPLAY and specify the unit where the tape is located. For example,

) TYPE @MTD1 ↵
...

or

) XEQ DISPLAY @MTD1 ↵
VOL1VOL001 ...
...

In the display, the volid immediately follows the VOL1 header (characters VOL1). In the preceding example, the volid is VOL001. Other fields, like the filename, are explained in the next section.

Components of Labeled Tape

There are several different kinds of labels written to a labeled tape. The most important ones are

- the volume header label (contains the volid and is created by the LABEL utility) and
- the first file header label (HDR1, created by the system when it writes material to the tape).

Generally, the other labels are useful only if you have application programs that specifically read and write labels, via the ?OPEN system call or high–level language equivalent.

A labeled tape, after it has been labeled via LABEL and files have been written to it, has the structure shown in Figure 6–1.

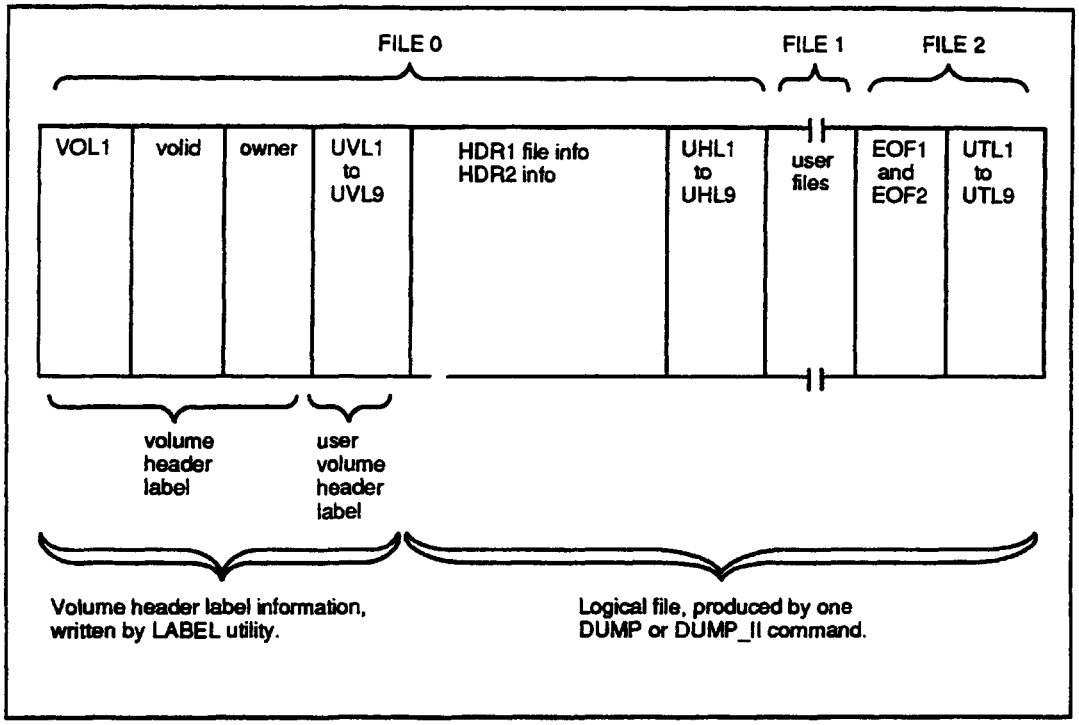


Figure 6–1 Information on a Labeled Tape

The fields shown on the tape in Figure 6–1 have the following meanings.

VOL1	A fixed string that means volume type 1. The LABEL utility always writes VOL1 on every tape.
volid	The volume ID assigned by a person using the LABEL utility (XEQ LABEL unitname volid). The reserved length is 6 characters; the volid cannot exceed 6 characters.
owner	The owner field (optional). Users may specify the contents with the LABEL /OWNER= switch. The maximum length is 14 characters for DG or ANSI label tapes, 10 characters for IBM label tapes. If you use the LABEL/OWNER= switch, you must also use the /OWNER= switch with DUMP_II. The DUMP command does not have a /OWNER switch, but retains the string from the LABEL utility.
UVL1 to UVL9	Volume labels that users specify with the LABEL /UVL=x switch. On the tape, each UVL (user volume label) field begins with the characters UVLn, followed by the text of x as given with the /UVL=xswitch. These labels are optional. They are useful only if your application programs can read them. The maximum length you can specify for a user volume label is 76 characters.

HDR1 file info

HDR1 is written by the operating system during the dump operation. It contains the following items.

- filename for the fileset, as supplied by the person who started the tape write. (This name has no relationship to individual files actually written to the tape.) The tape filename can be as many as 17 characters long. For example, the following command creates the filename FILE5 on the tape.

) DUMP_II/V TAPE:FILE5 ↵

- fileset ID;
- file section number;
- version number;
- generation number;
- creation date;
- creation date;
- expiration date (the default is 90 days from the creation date; the dump command /RETAIN switch can override the default);
- access to this tape file;
- block count (always 0 in HDR1);
- operating system ID (as set by SYSID command, or the default).

HDR2 info

In Data General format (default), the system always writes this label. In non–Data General format, it is written only if a user program opened the tape using an extended packet on the ?OPEN call. HDR2 contains the record format specifier (a one–letter code), block length (buffer size), and a code for record length.

UHL1 to UHL9

These are user header labels. They are optional. The only way to have them written is via the ?OPEN system call or higher–level language equivalent. They are useful only if you have application programs that read and write them.

user files

This is the disk–based information copied in the tape write operation. It includes all information written by any single CLI command or program write statement. For example, if the command is DUMP_II, user files will include all files dumped.

EOF1 and EOF2

At the end of each logical file written, the system writes a label with EOF1 and EOF2. It writes an EOV1 and EOV2 label at the end of each volume if another volume is required. EOF1 and EOV1 each record the number of blocks written; this number serves as an error check when the tape is read.

UTL1 to UTL9

Optional user trailer labels, written after the EOF or EOV label.

Data General, ANSI, or IBM Format?

By default, labeled tapes are written in Data General format (DG format is compatible with, but not identical to, ANSI format). To write labeled tapes that will be read on a Data General, ANSI, or IBM (EBCDIC) system, proceed as follows.

- If the destination system is one of the Data General operating systems AOS, AOS/VS, or AOS/VS II, use the LABEL program without the /I switch. To write to the tape, you can use any CLI command or the default format in any write statement.
- If the destination system is an ANSI–based, non–Data General system, use the LABEL program without the /I switch. To write to the tape, you can't use dump or load commands. You must use a program that either opens the labeled tape (?OPEN call to @LMT:volid:filename) for ANSI format; or use a program that employs the high–level language equivalent of ?OPEN/?WRITE to specify ANSI format.
- If the destination system is an IBM system (EBCDIC), use LABEL with the /I switch. You should then write to the tape using thee /IBM switch with the DUMP_II or DUMP command. On the IBM system, load the tape using the /IBM switch as well.

Requesting the Mounting of a Labeled Tape

Labeled tape requires intervention of the system operator, who must mount the volumes you request and tell EXEC that they are mounted. There are, however, two methods for requesting a labeled tape to be mounted: implicit and explicit. Each method is explained in the sections that follow.

Using an Explicit Mount Request

To explicitly request the operator to mount a labeled tape, use the MOUNT command (described in Chapter 5) with the /VOLID= switch.

When you issue a MOUNT request, you specify a *linkname,* which you will subsequently use to refer to the mounted tape. After mounting the tape, the operator tells the system which unit is being used. The system then creates your link file and associates it with that tape unit. Your linkname will then automatically refer to the appropriate unit. The link file is only temporary — it will be deleted when you dismount the tape — so the name you choose for it is not critical.

The system creates the link file in your initial directory unless you use the /DIRECTORY switch with the MOUNT command to specify another directory. Therefore, the linkname you specify in the MOUNT command cannot already exist in the destination directory. When you subsequently use this linkname in a DUMP_II/DUMP or LOAD_II/LOAD command, be sure the system can locate the link file. If the linkname is not in the directory from which you will issue the DUMP_II/DUMP/LOAD_II/LOAD command, put the linkname's directory in your search list or supply a full pathname in the DUMP_II/DUMP/LOAD_II/LOAD command.

You can include a /VOLID= switch for each volume you want mounted, and list them in the order you want the volumes mounted. The switch argument is the volume ID. For example,

```
) MOUNT/VOLID=VOL001/VOLID=VOL002/VOLID=VOL003 MYTAPE Please ↵
```

When dumping a large amount of data, you may not know exactly how many tape volumes you will need. In such a case, you can either specify more volume IDs than you will need (the system will use only what it needs), or you can use the /EXTEND switch with the MOUNT command. This switch lets your operation continue even if you specified too few volids on the MOUNT request. But /EXTEND does not specify the volume IDs of the additional volumes; this places the responsibility for choosing the additional labels and labeling the tape in the hands of the person at the system console.

If the dump operation requires more volumes than you specified, and you did not use /EXTEND, the dump may be invalid; if so, you will have to start over. In best case, with DUMP_II, the program will prompt you for the another volume, and you will find it awkward to ask the operator to label and mount another tape. So be generous with your volume IDs — or use the /EXTEND switch.

In the MOUNT command, you can include a message for the system operator. You can use this message to ask the operator to insert a write–enable ring, for example.

After you issue the MOUNT request, the operator will either mount the tape or refuse your request. If you did not use the /NOPEND switch on the MOUNT command, your CLI prompt will not reappear until the operator responds. You can cancel your MOUNT request by typing CTRL–C CTRL–A; you should then delete the link file from your initial directory.

When the first volume is mounted, you can proceed with your labeled tape operation. To refer to a file on a labeled tape, specify the linkname you designated in the MOUNT command, and append the name of the tape file (preceded by a colon); the filename can be a maximum of 17 characters in length. The DUMP_II/DUMP command creates the tape filename, so you can use any valid characters you want for it. Note that this means you cannot dump to an existing tape file without overwriting all information that currently exists in this file.

When you want to load the dumped information, you will need to remember the tape filename used for the dump. The format of the dump and load commands with labeled tape is

```
DUMP_II   }
DUMP      } linkname:tape–filename  [pathname–template] [...]
LOAD_II   }
LOAD      }
```

For example,

```
) DUMP_II/V  MYTAPE:REPORTS  REPORT+ ↵
```

Labeled Tape Example

To request the operator to mount a labeled tape, you could type

```
) MOUNT/NOPEND/VOLID=VOL1/VOLID=VOL2/EXTEND  NEWTAPE Ring in please ↵
```

After the tape is mounted, you use full pathname to the linkname you specified in the MOUNT command, NEWTAPE.

```
) DUMP_II/V/RETAIN=30  :UDD:JR:NEWTAPE:SOURCES  +.SR ↵
.
(Displays files dumped)
.
) DUMP_II/V/RETAIN=30  :UDD:JR:NEWTAPE:LISTINGS  +.LS ↵
.
(Displays files dumped)
.
```

The first command creates a tape file named SOURCES and dumps all files ending with .SR in the working directory to file SOURCES. The next command creates tape file LISTINGS and dumps files ending in .LS to that tape file. The /RETAIN switch specifies a retention period of 30 days. Also, notice that the linkname, NEWFILE, is given as a full pathname to ensure that the program can find it.

When you are finished with the tape, use the DISMOUNT command to request the operator to remove the tape from the unit.

```
) DISMOUNT NEWTAPE Thanks – please place in rack ↵
```

(Instead of using DISMOUNT, you can use QCANCEL and specify the sequence number of the job in the mount queue. To display the contents of the mount queue, use the command QDISPLAY/TYPE=MOUNT or use the number displayed after you enter the MOUNT command.)

Using an Implicit Mount Request

You can work with labeled tapes without explicitly asking the operator to mount a tape. This method is most useful in a situation where the single volume ID you specify enables the operator to locate the correct tape(s). In other words, since you do not use the MOUNT command, you can specify only the first volume ID, and you do not have the option of sending the operator a brief message.

With the implicit method, you specify the device name @LMT (which stands for Labeled Magnetic Tape), the volume ID, and the filename of the tape file. The tape filename can be as many as 17 characters long. On a dump, the system tries to create a file with this name on the tape; on a load, you must specify the name of an existing file. The format is

```
@LMT:volid:tape–filename
```

For example, if you wanted to load the contents of a tape file called SOURCES on volume VOL1 (as dumped to above), you might type

```
) LOAD_II/V @LMT:VOL1:SOURCES ↵
```

The system sends a message to the operator console, requesting VOL1 to be mounted. The operator can either mount the volume or refuse the request. Your LOAD_II command will execute after the operator mounts the tape and tells the system which unit to use. The CLI prompt returns after the load completes.

If your request complete normally, the system will prompt the operator to dismount the tape. If you cancel or interrupt the request, you must use the DISMOUNT command to prompt the operator to dismount the tape.

Implicit Mount Example

To dump all files in your initial directory to a labeled tape whose volume ID is VOL024, you could use the command:

```
) DIRECTORY/I ↵
) DUMP_II/V/L=INITIAL.DUMPLIST @LMT:VOL024:INITIAL ↵
```

The operator receives a message asking for VOL024 to be mounted. He or she can then search for this tape volume (or label a tape with the volume ID VOL024), and then mount this tape and issue a command that indicates which unit the tape is mounted on. Your DUMP_II command then executes. The CLI prompt will return after the dump is completed; the system requests the operator to dismount the tape.

If the operator decided not to mount the tape, he or she would type a *refused* command to EXEC. You would see the message

Warning: Refused by operator, File @LMT:VOL024:INITIAL

An operator might refuse a mount request if no tape unit were available, or if he or she did not want to search for the specified volume.

Specifying a Retention Period for a Dump

Every label specifies a retention period for the data in that tape file. The system will return an error message to any attempt to overwrite files whose retention period has not expired.

To specify a retention period, use the /RETAIN= switch on your DUMP or DUMP_II command. Specify the number of days in the retention period. If you specify 0, you can reuse the tape immediately. If you omit the switch, the system automatically assigns a default retention period of 90 days.

You can override a volume's retention period by overwriting its label. Doing so, however, makes all data on that volume unavailable. You should use this method only if you no longer need the information on the volume.

Dumping to or Loading from Specific Volumes

All material written by a write operation (for example, by one DUMP_II or COPY command), makes up a single logical file. The logical file may fit on one volume or span many volumes. To read any part of this logical file, the system must read sequentially from the beginning — through multiple tape volumes if needed — until it reaches the desired tape file.

There is a way to add a logical file to the last file in a tape set, or to read from a logical file that starts on a tape other than the first tape in the fileset — without reading all preceding tape volumes. This technique works only if you do multiple writes (like multiple dumps) to a fileset that spans multiple volumes. The dump and load commands and utilities have a /SPECIFIC switch that tells the program to start at a specific volume ID. For example, assume the commands

```
) DUMP/II/V XXXTAPE:TEXT_BACKUP TEXTFILES:# ↵
```

The first dump (logical file) consumes one tape volume, volid VOL1, and part of a second volume, volid VOL2. Now (or in future) if you want to append another logical file to the end of the first, you could use the /SPECIFIC switch in an implicit mount with the second volume, as follows.

```
) DUMP_II/V/SPECIFIC @LMT:VOL2:SOURCES +.SR ↵
```

Someone mounts VOL2 and tells EXEC it is mounted. The system then spins tape VOL2 to the end of the preceding logical file, TEXT_BACKUP, creates a logical file named SOURCES, and starts dumping all +.SR files to that logical file. After it dumps the +.SR files, it rewinds the tape.

Loading a file from a specific volume works the same way. You specify an implicit labeled mount, with @LMT:volid:tape–filename. After soneone mounts the tape and tells EXEC it is mounted, the system will spin through the volume, looking for the tape file. If it finds the tape–file, it will load from it. For example, to load from the VOL2 made above, you might type

```
) LOAD_II/V/RECENT/SPECIFIC @LMT:VOL2:SOURCES ↵
```

There are limitations with the /SPECIFIC arrangement; to be useful, it requires different logical files on one tape set (multiple dumps per tape set), and it requires you to keep good records of the dump(s) on each volume. The switch does not provide disk file indexing by volume, which is available with the optional DUMP_3/LOAD_3 utilities.

Using Labeled Tape with User Programs

A user program can gain access to a labeled tape in one of two ways.

1. If you mount the tape (via the MOUNT command) before running the program, the program can open the file linkname:filename to gain access to the tape. If the program requires multiple volumes, this method is preferred.
2. The program can open the file @LMT:volid:filename, which will generate an implicit mount request.

In either case, the program must open the tape file for *either* input or output, but not both. The program can write or read sequential, fixed–length, or variable–length records only; it cannot write or read data–sensitive or dynamic length records.

If you want to create your own labels rather than using the LABEL utility, refer to the standard label formats described in your system programmer's manual.

Using Labeled Tapes in Batch Mode

Batch mode is ideal for lengthy dump or load tape operations. Batch mode frees your terminal for other activity, and allows you to queue a job to run at a time when the system may not be so heavily used.

A batch request for a job that will use labeled tape should include the /OPERATOR switch to ensure that an operator is available to handle the required tape volumes. Then, if the operator is not on duty, the system will not start the job. See the QBATCH command in Chapter 5.

Acting as System Operator

This section explains what you must do to respond to MOUNT and DISMOUNT requests sent to the system operator. If you are the system operator, or find that you must perform system operator functions, you should read this section.

NOTE: To issue a MOUNT request, you must be logged on under EXEC. The CLI operator process, PID 2, and its sons cannot cannot issue a MOUNT request.

Coming on Duty

Mount requests are displayed on the operator console only when an operator is on duty. To inform the system that you are available to respond to mount requests, enter this command at the operator console:

```
) CONTROL @EXEC OPERATOR ON ↵
From Pid   3 : (EXEC) System Operator On Duty at @CON0
```

If you want to assign the system operator to a specific console, include the console's device name after the ON argument. For example

```
) CONTROL @EXEC OPERATOR ON @CON6 ↵
From Pid   3 : (EXEC) System Operator On Duty at @CON6
```

Complying with a Mount Request

When a user issues a MOUNT request, a message similar to the following appears on the system console:

```
From PID 5: (XMNT) 29-Oct-1991 16:24:04
                   ********** **
                   Labeled Mount Request
                   ********************
                   MID=      i,          USER=xxx
                   USER PID=j            Requestor PID =k
                   Volumes:  VOLvvv0, VOLvvv1

From PID 5:
                   Mount Volume          VOLvvv0
                   Settings:             Default Density
                   User Message:         Please Use Large Tapes

                   Respond:              CX MOUNTED [@unitname]
                             or          CX REFUSED
```

An unlabeled tape mount request is identical, except there is no *Volumes*, *Mount Volume*, or *User Message* line.

To comply with the request, mount the specified tape volume, with volume ID vvv0, on an available unit. Then, at the system console, use a MOUNTED command in the following form:

```
CONTROL @EXEC MOUNTED @unitname
```

where unitname is the device name of the tape unit where you mounted the tape.

Refusing a Mount Request

If you must refuse the MOUNT request, respond with this command:

```
) CONTROL @EXEC REFUSED )
```

The following message will appear on the user's terminal:

Warning: Refused by operator ...

You may have to refuse a mount request if you cannot find the requested volume, if that volume is already being used by someone else, or if there is no available tape unit. It's a good idea to use the SEND command to explain your reason for refusal to the person who asked for the mount. His or her PID is displayed in EXEC's mount request message.

Dismounting a Tape

When finished with a mounted tape, a user should either issue a DISMOUNT request or cancel the entry in the mount queue (via QCANCEL). A message similar to the following then appears on the system console:

From PID 5: (XMNT) 29–Oct–1991 16:24:04
*********** ** *
Unit Dismount Request
****
MID= i, USER=xxx
USER PID=j Requestor PID =k
Volumes: VOLvvv0
Units: @MTB0 / VOLvvv0

From PID 5:
Dismount Unit @MTB0
User Message: Please Mark the Tape with my name.

Respond: CX DISMOUNTED[@unitname]

Remove the tape from the unit, and then at the system console type

) CONTROL @EXEC DISMOUNTED ↵

If additional volumes are needed, EXEC will prompt for each one. You should mount them and respond CONTROL @EXEC MOUNTED as described earlier.

Summary of Tape Operations

Figure 6–2 summarizes the procedures for using unlabeled and labeled magnetic tape.

Unlabeled Tape — No System Operator

1. Mount your tape on a tape unit.
2. Specify the device name and tape file number for a dump or load.

) DUMP_II/V @MTD0:1 +.F77 ↵
3. Rewind the tape.

) REWIND @MTD0 ↵
4. Dismount and store the tape.

Unlabeled Tape — via System Operator

1. Use MOUNT command to ask that tape be mounted; specify a linkname. To have prompt return immediately, use /NOPEND switch.

) MOUNT MYTAPE Please insert write–enable ring ↵
2. Specify the linkname and tape file number for a dump or load:

)DUMP_II/V MYTAPE:2 +.LS ↵
3. Request operator to dismount tape:

) DISMOUNT MYTAPE ↵

Labeled Tape — Explicit Mount Request

1. Use MOUNT command to ask that tape be mounted; specify a linkname. To have prompt return immediately, use /NOPEND switch.

) MOUNT/VOLID=V1/VOLID=V2 MYTAPE Please insert ring ↵
2. Specify the linkname and tape filename for a dump or load.

) DUMP_II/V MYTAPE:PROGRAMS +.PR ↵
3. Request operator to dismount tape.

) DISMOUNT MYTAPE ↵

Labeled Tape, Operator on Duty — Implicit Mount Request

1. Specify the device @LMT, the volume ID (volid), and tape filename for a dump or load.

) DUMP_II/V @LMT:JOAN:PROGRAMS +.PR ↵

Figure 6–2 Summary of Magnetic Tape Operations

About Diskettes

A diskette (also known as a *floppy disk*) is another medium you can use to store information off line.

Diskettes may be labeled or unlabeled. Labeled diskettes let you:

- store information that exceeds the capacity of a single diskette;
- uniquely identify diskettes, to ensure that you are loading the correct diskette set;
- protect your data from being overwritten by specifying a retention period.

Labeled diskettes are not supported by CLI32. To use labell=ed diskettes, you must use CLI16, the 16–bit CLI.

Using Unlabeled Diskettes

To use an unlabeled diskette, place the diskette in a unit. When you issue a DUMP_II/DUMP or LOAD_II/LOAD command, for example, specify the diskette unit device name and the file(s) you want to dump or load.

The default device name for a diskette unit is @DPJ10.

The following command dumps the file MYFILE to the diskette inserted in the default unit (@DPJ10).

```
) DUMP_II/V @DPJ10 MYFILE ↵
```

You use the same format to load file(s) from a diskette. For example,

```
) LOAD_II/V @DPJ10 +.PR ↵
```

This command loads into the working directory all program files (.PR) on the diskette located in unit @DPJ10. The diskette must have been dumped to by DUMP_II or DUMP.

Using Labeled Diskettes (DUMP Command, CLI16 only)

Labeled diskettes are supported only by the DUMP and LOAD commands. Since CLI32 does not include these commands, you cannot use labeled diskettes with CLI32. To use labeled diskettes, you must use CLI16.

Before you can use labeled diskettes, you must turn on CLI operator mode via the OPERATOR command. This command and mode are unrelated to the EXEC OPERATOR command. Type

```
) OPERATOR ON ↵
```

If you try to use labeled diskettes when operator mode is not turned on, the system displays this message in response to your request:

Warning: No operator available

To refer to a labeled diskette, use the device name @LFD (which stands for Labeled Floppy Disk), followed by the volume ID (volid) and filename. This is the format

```
@LFD:volid:filename
```

The system prompts you to insert the appropriate volume in a specific diskette unit.

Please insert a diskette if not already inserted
Unit [@DPJ10] Volume ID [volid] [y] ?

The default response is Y, so you can continue simply by pressing the NEW LINE key. If the unit or volid is incorrect, you can cancel the operation by typing N and pressing the NEW LINE key.

Labeling a Diskette

The easiest way to label an unlabeled diskette is to have the CLI do it. To do this, you must turn on automatic labeling. Do this by including the /LABEL switch when turning on operator mode:

```
) OPERATOR/LABEL ON ↵
```

If operator mode is already on, but labeling is off, first turn off operator mode, and then issue the command just shown.

When automatic labeling is on, the system will label (or relabel) the first diskette with the volid you specify in the DUMP command. It will create volume IDs by adding 1 to the volume ID you specify. Therefore, you should specify a volume ID that ends with a number. For example, if you specify VOL001, the first diskette will be labeled VOL001, the second (if needed) VOL002, the third (if needed) VOL003, and so on.

Automatic labeling overrides the system's normal label–checking procedure. Use it only if you are dumping data to an unlabeled diskette, or if you want to replace the contents of a previously labeled diskette.

If you want the system to check a diskette label before writing new information to it, be sure automatic labeling is off. To turn off automatic labeling, turn operator mode off, and then turn it back on without the /LABEL switch:

```
) OPERATOR OFF ↵
) OPERATOR ON ↵
```

When automatic labeling is off and you write file(s) to a diskette, the system checks the diskette for the volume ID (volid) you specify. If the diskette has no volid (i.e., is unlabeled) or a volid other than the one you specified, the system displays the message:

The label on the diskette is not the label requested
Inserted: volid_x Requested: volid_y

Do you want to relabel this diskette? [n]

To label the diskette with the volid you specified, type Y and then press the NEW LINE key. (A note of caution: if you relabel the diskette, all existing data on the diskette is lost.) Otherwise, enter N or press the NEW LINE key.

Diskette Examples

```
) OPERATOR/LABEL ON ↵
) OPERATOR ↵
ON, Labeling ON
```

The first command turns on operator mode and automatic labeling, as the second command confirms.

```
) DUMP/V/RETAIN=0 @LFD:AL0001:ALLFILES ↵

Please insert a diskette if not already inserted
Unit [@DPJ10] Volume ID [AL0001] ? [y]

(Insert diskette and press NEW LINE.)
(System dumps files to diskette.)
.

Please insert next diskette if not already inserted
Unit [@DPJ10] Volume ID [AL0002] ? [y]

(Remove diskette; insert next diskette and press NEW LINE.)
(System dumps files to diskette.)
.

) OPERATOR OFF ↵
```

The DUMP command uses the device name @LFD to indicate a labeled diskette operation. The command specifies the volume ID (volid) AL0001 and the filename ALLFILES, which will hold all the files dumped from the working directory. The /RETAIN= switch sets the retention period to 0 days, which allows the diskette to be reused immediately.

The system prompts the user to insert the first volume (AL0001) in the default unit (@DPJ10). When the capacity of the first volume is filled, the system prompts the user to insert the next volume (AL0002).

Legal volume IDs are from one to six printable characters. The CLI generates subsequent volids based on the first one.

- If the last character is numeric, the system increments it by one for each subsequent volume.
- If the last character is not numeric, the system adds the digit 1 to the first subsequent volume, and then increments it with each additional volume.

NOTE: If incrementing the volid would exceed the naming limit of six characters, the new number replaces the first character to its left. For example, the volume after MYVOL9 is MYVO10.

The next example shows a load from labeled diskette.

```
) OPERATOR ON ↵
 ) OPERATOR ↵
 ON, Labeling OFF

) LOAD/V/RECENT  @LFD:AL0001:ALLFILES ↵

Please insert a diskette if not already inserted
Unit [@DPJ10] Volume ID [AL0001] ? [y]
```

(Insert correct diskette and press NEW LINE.) .

(System loads files from diskette, then prompts for the next; you insert it and press NEW LINE.)

```
) OPERATOR OFF ↵
```

The first OPERATOR command omits the /LABEL switch so that automatic labeling is not turned on, as confirmed by the second command.

The LOAD command uses the device name @LFD to indicate a labeled diskette operation. This command specifies the volume ID (AL0001), and the name of the diskette file (ALLFILES) whose contents are to be loaded into the working directory.

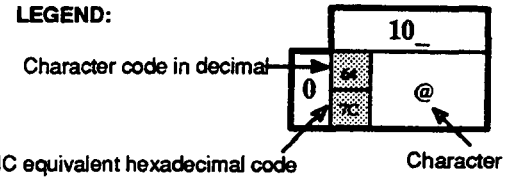
End of Chapter

Appendix A
ASCII Code Set

To find the octal value of a character, locate the character, and combine the first two digits at the top of the character's column with the third digit in the far left column. For example, the octal value of the character & is 046.

↑ means CONTROL

LEGEND:



OCTAL	00_	01_	02_	03_	04_	05_	06_	07_
0	0 00 NUL	8 08 BS (BACK SPACE)	16 10 DLE ↑P	24 18 CAN X	32 20 SPACE	40 28 (	48 30 0	56 38 8
1	1 01 SOH ↑A	9 09 HT (TAB)	17 11 DC1 ↑Q	25 19 EM ↑Y	33 21 !	41 29)	49 31 1	57 39 9
2	2 02 STX ↑B	10 0A NL (NEW LINE)	18 12 DC2 ↑R	26 1A SUB ↑Z	34 22 " (QUOTE)	42 2A *	50 32 2	58 3A : (COLON)
3	3 03 ETX ↑C	11 0B VT (VERT TAB)	19 13 DC3 ↑S	27 1B ESC (ESCAPE)	35 23 #	43 2B +	51 33 3	59 3B ; (SEMI-COLON)
4	4 04 EOT ↑D	12 0C FF (FORM FEED)	20 14 DC4 ↑T	28 1C FS ↑\	36 24 $	44 2C , (COMMA)	52 34 4	60 3C <
5	5 05 ENQ ↑E	13 0D RT (RETURN)	21 15 NAK ↑U	29 1D GS ↑]	37 25 %	45 2D -	53 35 5	61 3D =
6	6 06 ACK ↑F	14 0E SO ↑N	22 16 SYN ↑V	30 1E RS ↑↑	38 26 &	46 2E . (PERIOD)	54 36 6	62 3E >
7	7 07 BEL ↑G	15 0F SI ↑O	23 17 ETB ↑W	31 1F US ↑←	39 27 ' (RIGHT APOSTROPHE)	47 2F /	55 37 7	63 3F ?

OCTAL	10_	11_	12_	13_	14_	15_	16_	17_
0	64 40 7C @	72 48 C8 H	80 50 D7 P	88 58 E7 X	96 60 79 ` (GRAVE or LEFT APOSTROPHE)	104 68 88 h	112 70 97 p	120 78 A7 x
1	65 41 C1 A	73 49 C9 I	81 51 D8 Q	89 59 E8 Y	97 61 81 a	105 69 89 i	113 71 98 q	121 79 A8 y
2	66 42 C2 B	74 4A D1 J	82 52 D9 R	90 5A E9 Z	98 62 82 b	106 6A 91 j	114 72 99 r	122 7A A9 z
3	67 43 C3 C	75 4B D2 K	83 53 E2 S	91 5B 8D [	99 63 83 c	107 6B 92 k	115 73 A2 s	123 7B C0 {
4	68 44 C4 D	76 4C D3 L	84 54 E3 T	92 5C E0 \	100 64 84 d	108 6C 93 l	116 74 A3 t	124 7C 4F |
5	69 45 C5 E	77 4D D4 M	85 55 E4 U	93 5D 9D]	101 65 85 e	109 6D 94 m	117 75 A4 u	125 7D D0 }
6	70 46 C6 F	78 4E D5 N	86 56 E5 V	94 5E 5F ↑ or ^	102 66 86 f	110 6E 95 n	118 76 A5 v	126 7E A1 ~ (TILDE)
7	71 47 C7 G	79 4F D6 O	87 57 E6 W	95 5F 6D ← or _	103 67 87 g	111 6F 96 o	119 77 A6 w	127 7F 07 DEL (RUBOUT)

End of Appendix

Appendix B
Keyboard Summary

Your terminal may be a DASHER video display or a hardcopy terminal. In either case, it will have a keyboard, which you use to issue CLI commands.

The keyboard includes *alphanumeric* keys, which you press to enter an alphabetic, numeric, or other symbol, and *function* keys, which you press to perform an operation. The next several sections describe the keys on your keyboard, and explain how they apply to the CLI.

Numeric Keypad

You can use both the numeric keypad at the far right of your keyboard or the numeric keys on the main keypad to enter digits.

Cursor Control Keypad (Video Display Terminals Only)

Video display terminals use a *cursor* to mark your place on the screen and to show where your keyboard input will appear. To the left of the numeric keypad is the cursor control keypad; it contains keys that you can use to move the cursor around the screen.

Key Summary

Next is a summary of the special keys that may be on your terminal.

↑	(Uparrow or cursor up key) Within some system utilities, moves the cursor up one line while remaining in the same column.
↓	(Downarrow or cursor down key) Within some system utilities, moves the cursor down one line while remaining in the same column.
←	(Leftarrow or cursor left key) Moves the cursor to the left one character position (column).
→	(Rightarrow or cursor right key) Moves the cursor to the right one character position (column).
ALPHA LOCK	Turns ALPHA LOCK on or off. When it is on, alphabetic keys generate uppercase letters without your having to use the SHIFT key.
BREAK/ESC	Puts your terminal back in text mode. If you display an unprintable file, your terminal may be put into binary mode, which prevents it from recognizing a console interrupt sequence. Press CMD, then BREAK/ESC.
CMD	(Command) Generates a command character when pressed with another key. There isn't any difference between uppercase and lowercase letters when used with this key.
CR	(Carriage Return) Enters the current command line but ignores any characters to the right of the cursor. If you are typing text into a file, this key moves the cursor to the beginning of the next line.
CURSR TYPE	(Cursor Type) Changes the appearance of the cursor.
CTRL	(Control) Generates a control character when pressed with another key. There isn't any difference between uppercase and lowercase letters when used with this key.
DEL	(Delete — some keyboards call this key RUBOUT.) On video displays: deletes the character to the left of the cursor, moving the cursor and the rest of the line one position to the left. On hardcopy terminals: deletes a character from the end of the current line. Each time you delete a character with this key, a backarrow or underscore appears.
ERASE EOL	(Erase End–of–Line) Erases characters, beginning with the current cursor position, to the end of the line.
ERASE PAGE	Clears the screen of text and moves the cursor to the top left corner of the screen. It also enters any characters that are on the current line. If the display and keyboard seem to freeze after you press ERASE PAGE, you may have page mode turned on; to free the display and keyboard, press the CTRL key and the Q key at the same time.

HOLD	Alternately freezes and releases the screen image. This key is to temporarily stop text that is scrolling; it is often easier to use than the CTRL–Q sequence.
HOME	Moves the cursor to the left margin of the current line.
NEW LINE	Enters the current command line. If you are typing text into a file, this key moves the cursor to the beginning of the next line.
NORM COMP	(Normal/Compress) Alternately changes the display to either normal or compressed spacing for characters.
ON LINE	When pressed with the CMD key, alternately puts the terminal on or off line.
REPT	(Repeat) Repeats the key that you press with it as long as you hold both keys down.
RUBOUT	(Some keyboards call this key DEL.) On video displays: deletes the character to the left of the cursor, moving the cursor and the rest of the line one position to the left. On hardcopy terminals: deletes the character at the end of the current line. Each time you delete a character with this key, a backarrow or underscore appears.
SCRLL RATE	(Scroll Rate) Changes the terminal's rate for scrolling text.
SHIFT	Generates the Shift value (if one exists) for the key you press at the same time.
SPCL	(Special) Generates a special character when pressed with certain other keys.
TAB	Moves the cursor to the next tab stop and inserts a tab character in the line. In the CLI, a tab occupies every 8 character positions, so tab stops are in columns 9, 17, 25, 33, and so on. Other software, such as a text editor, may use this key differently; see the software manual for any information about the TAB key.

End of Appendix

Appendix C
Submitting Batch Jobs in Stacked Format

This appendix applies only to AOS/VS, since AOS/VS II does not support card readers.

When you submit batch jobs to a card reader, they must be in a special *stacked* format. You do not use any CLI commands to submit a job in stacked format, but your job should contain all the CLI commands it needs for processing. Simply place the job in the card reader or give them to the operator.

All stacked batch jobs must have the format illustrated in Figure C–1.

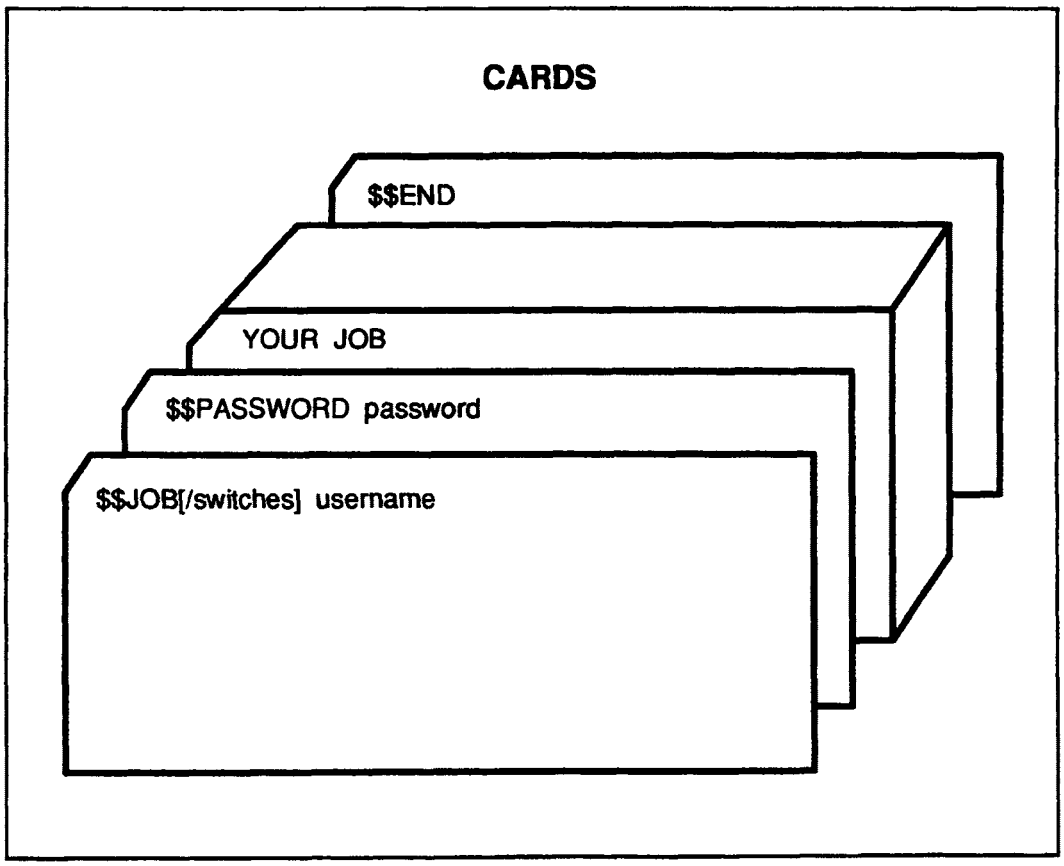


Figure C–1 Stacked Format for Batch Jobs

Each job must start with a job control card whose format is

```
$$JOB[/switches] username
```

Table C–1 contains a list and description of batch job switches. Control cards accept unique abbreviations. For example, $$J is equivalent to $$JOB.

Follow the $$JOB card with a password card whose format is

```
$$PASSWORD yourpassword
```

For security reasons, protect your password card. After the $$PASSWORD card, enter your job. Batch jobs from an actual card reader will contain 80 ASCII characters on each card. AOS/VS appends a NEW LINE character to each card image. If a card contains fewer than 80 characters, AOS/VS retains the trailing spaces.

The last card in your stacked job must be an end card whose format is

```
$$END
```

If your job contains any $$ entries other than those shown above, AOS/VS will not queue or process the job.

You must include an end of file card to turn off the card reader. An end of file card has all the holes punched in the first column. For more information, see *Installing, Starting, and Stopping AOS/VS*.

Switches you can use when submitted a job are explained in Table C–1.

Table C–1 Optional Batch Job Switches

Switch	Effect
/CPU=hh*[:mm[:ss]]*	Limit the amount of CPU time this batch job can use. The CLI requires the hours (hh). Minutes (mm) and seconds (ss) are optional. The maximum time is 36:24:32. The /CPU switch takes effect only if you have enabled processing limiting via the command CONTROL @EXEC LIMIT
/AFTER=*[[date:]time]*	Delay processing until a future time. If you specify no date, AOS/VS uses today's date. If you specify no time, AOS/VS uses midnight. The date is in the form dd–mmm–yy (for example, 08–SEP–89 for September 8, 1989.) You must specify all three parts of the date. Time is in the form hh[:mm[:ss]]. The CLI requires hours (hh) while minutes (mm) and seconds (ss) are optional. Specify the time on a 24–hour system. For example, to begin processing a job after 3 p.m. on September 7, 1990, user SALLY could create a card containing $$JOB/AFTER=07–SEP–90:14 SALLY
/AFTER=+time	Delay processing until a specific amount of time has elapsed. Time is in the form hh*[:mm[:ss]]*. The CLI requires hours (hh) while minutes (mm) and seconds (ss) are optional. For example, to begin processing a job in one week, create a card containing $$JOB/AFTER=+168 username
/HOLD	Do not process this job until the user issues a QUNHOLD command that identifies the job by its sequence number.
/JOBNAME	You must give this job a jobname before you can issue HOLD and UNHOLD commands against it by its name. (You can also issue HOLD and UNHOLD commands against a job according to its sequence number.)
/NORESTART	If AOS/VS or EXEC fails while it is processing this job, do not restart it.

(continued)

Table C–1 Optional Batch Job Switches

Switch	Effect
/OPERATOR	This job requires an operator to be on duty. Use this switch if your job requires an operator to mount tapes, dismount tapes, install special forms on a printer, etc.
/QPRIORITY=qpriority	Specify a queuing priority for this job. It must be smaller than or equal to the queuing priority in the profile belonging to username. The highest priority is 1; the lowest is 255. If you omit this switch, the default value of qpriority is halfway between the limit in the user profile and the lowest priority, 255.
/QUEUE=queuename	Specify the name of the queue that will receive this job entry. If you omit this switch, the default queue is BATCH_INPUT.

(concluded)

Between the $$PASSWORD and $$END cards, you may place any CLI command lines. The system places these statements in a disk file, and places an entry for the file on the BATCH_INPUT queue. When job processing begins, the initial CLI environment is as follows:

- The LISTFILE is equivalent to the line printer.
- The @INPUT file is equivalent to the file containing CLI command lines.
- The DATAFILE is set to NULL.
- CLASS1 is set to ABORT.
- All other settings are the normal default values.

Figure C–2 illustrates a card file that you can submit to the batch facility. It assembles, links, and executes an assembly language program. Next, it lists the filenames in the initial working directory. When the batch job completes, the system deletes the file that contains the images of the cards between the $$PASSWORD and $$END cards.

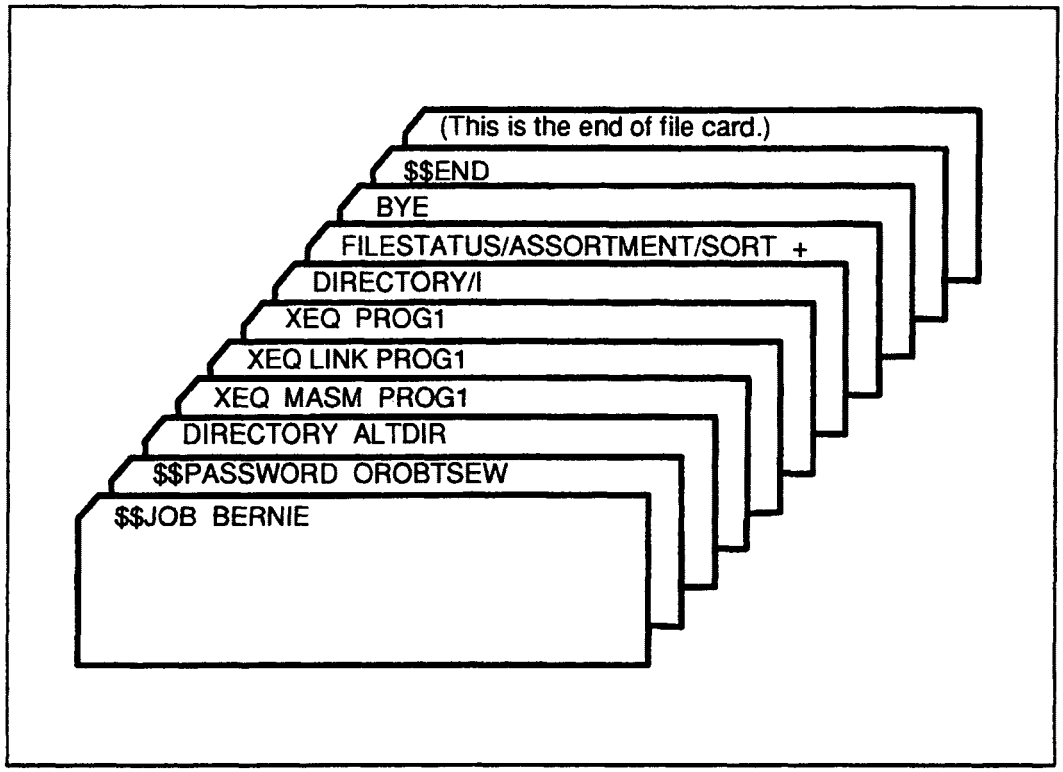


Figure C–2 Sample Batch Job in Stacked Format

End of Appendix

Appendix D
VT100 Support

This appendix explains how AOS/VS and AOS/VS II support VT100–compatible terminals and VT100 terminal emulation. It describes how to set up a VT100–class terminal for use with AOS/VS or AOS/VS II. The major sections proceed as follows.

- About VT100 Support
- Settting Up VT100 Terminal Hardware
- Setting Console Characteristics
- D200 Input and VT100 Output
- Simulating DASHER D200 Function Keys
- Example Session
- Using a Template

About VT100 Support

AOS/VS and AOS/VS II support VT100–compatible terminals on most asynchronous controllers. This support became effective with Revision 7.65 of AOS/VS and Release 1.10 of AOS/VS II, and includes VT100 terminal emulation such as the "xterm" program available in the X Window System. You enable the support through a console characteristic, /XLT, that you can specify using VSGEN or as part of a CHARACTERISTICS command.

AOS/VS and AOS/VS II translates some VT100 and VT220 input character sequences, such as cursor keys, function keys, and returned device status into comparable DASHER D200 input. On output, VT100 support recognizes DASHER D200 commands and translates them into comparable VT100 commands.

In this appendix, the NEW LINE symbol (↲) refers to the VT100 or VT220 terminal Carriage Return key. Its actual label can be RETURN or Return. In addition, *VT100 terminal* refers to both VT100 and VT220 terminals unless we specify otherwise; it may also refer to a program providing VT100 terminal emulation (such as the X Window System's *xterm*).

NOTE: If you use TELNET to call from an xterm window to an AOS/VS II system which supports Ring 0 TELNET, a properly enabled TCON supplies D200 emulation so that your TELNET session automatically supports appropriate cursor terminal support for AOS/VS programs. However, if you intend to use function–based applications (the CEO system, for example), we recommend using *mterm* emulation, since it provides mapping for function keys that do not work withVT100 emulation.

Terminal Controllers Supported

VSGEN (both AOS/VS II and AOS/VS versions) automatically enables VT100 support for all asynchronous controllers specified to VSGEN. If you want to change VT100 support, you specify it to VSGEN when you identify the controller, in the following query:

VT100 terminal support? (Y/N)

Respond Y if you want VT100 support; otherwise, respond N.

VSGEN VT100 support is not available on ATI devices. The support is available on all other controllers.

With either AOS/VS or AOS/VS II, you can attach a VT100 terminal to an IAC–16 controller. However, the IAC–16 controller does not have enough local memory to hold the code and data required for VT100 translation on all 16 of its lines. The IAC–16 with VT100 translation supports only 10 lines with the default I/O buffer size of 128 bytes. It supports 11 or 12 or 13 lines if you specify small enough I/O buffers. It does not support 14 or 15 or 16 lines. If you specify too many lines or I/O buffers that are too large, during system generation, then at startup the operating system will display an error message about IAC memory being oversubscribed. This IAC–16 will remain unusable after system startup. See the latest Release or Update Notice for your operating system for more information about I/O buffer size requirements.

Setting Up Terminal Hardware

To set up your VT100 or VT200 terminal for use with an AOS/VS or AOS/VS II system, begin by pressing your terminal's setup key to view or change terminal's hardware characteristics. The options are listed below. After viewing or changing options, press the setup key again. For more information, consult the manual for your VT100 or VT220 terminal.

Mode — Set the mode to either

- VT100
- VT220, 7 bit

XOFF — On a VT100, turn off AUTO–XOFF if you want to use page mode. On a VT220, select the NO XOFF option if you want to use page mode. (In Chapter 5, the description of the CHARACTERISTICS switch /PM explains page mode.)

A VT100 with AUTO–XOFF on, or a VT220 with NO XOFF off, will not let you generate an X–ON character from the keyboard. Since page mode holds output until you type the X–ON character CTRL–Q, page mode is useless with these two settings.

WRAP Turn on WRAPAROUND. (On a VT220, select AUTOWRAP.)

Some applications, such as CEO, take advantage of the DASHER D200 line wrap capability. These applications expect the cursor to wrap automatically to the start of the next line after reaching the last column of the current line.

Be aware that there are hardware differences between the ways DASHER D200 and VT100 terminals wrap the cursor to the next line. This may result in differences between the ways some applications work.

Setting CLI Console Characteristics

Use the following switches, as part of a CLI CHARACTERISTICS command, to specify the behavior of your VT100 terminal.

/XLT This characteristic enables VT100 support. The translation between a VT100 environment and a DASHER D200 environment takes place at all times, even on binary I/O. Do *not* use this characteristic for transferring data (for example, moving a file) over an asynchronous line.

/OTT Translates either the tilde (~) or the right brace (}) into an ESC character. Some Data General applications use the ESC character. These applications include the SED and SPEED text editors, CEO, and Business BASIC. A VT100 terminal has an ESC key, but a VT220 terminal does not have this key.

On a VT220 or other terminal that lacks an ESC key, you can set the /OTT characteristic. Setting /OTT has the system translate the two characters tilde (~, ASCII 176) and right brace (}, ASCII 175) to ESC. If /XLT is on, the /OTT–specified translation occurs even during binary read requests. With /OTT on, typing tilde or right brace is like typing the ESC key.

Setting /OTT, however, prevents you from using either a tilde or right brace in text, since the system translates them to ESC characters. If you need to use them in text (for example, for a C program, in which the right brace is essential), you can use the sequence CTRL–[(press the CTRL and left bracket keys together). CTRL–[generates an ESC without requiring you to set the /OTT characteristic.

To summarize: if you want to use a VT220 terminal in VT220 mode and you need to use the ESC key, you can set /OTT from the CLI and then use tilde or right brace in the program that needs ESC; *or* you can type CTRL–[in the program that needs ESC. Use the CTRL–[method if you need to use the tilde or right brace characters in text.

/KVT This characteristic enables Kanji VT100 support when used in conjunction with /XLT.

D200 Input and VT100 Output

On output from the operating system or your application program, VT100 support software recognizes D200 commands and translates them into equivalent VT100 commands. The software sends any other output directly to the terminal without translating it. This includes standard VT100 commands.

If your application sends a standard VT100 command, VT100 support sends it intact to your terminal. For example, suppose your application sets the top and bottom margins of your VT220 terminal's scrolling region by sending the seven characters

ESC [3 ; 2 0 r

The VT100 support software sends these characters to your VT220 terminal with no translation or alteration, regardless of the setting of characteristic /XLT.

When it receives input from your terminal, VT100 support software recognizes VT100 commands and translates them into equivalent D200 commands. It sends any other input directly to the operating system or your application program without translation or alteration, regardless of the setting of characteristic /XLT. This other input specifically includes native D200 commands. For example, pressing CTRL–H or CTRL–L on a VT100 terminal has the same effect as pressing the HOME key or the ERASE PAGE key, respectively, on a D200 terminal. Other VT100/D200 respective examples: the uparrow cursor control key and CTRL–W, the rightarrow cursor control key and CTRL–X, the leftarrow cursor control key and CTRL–Y, and the downarrow cursor control key and CTRL–Z.

Input Flow Control

A DASHER D200 terminal does not recognize CTRL–Q and CTRL–S as XON/XOFF flow control characters. In fact, it interprets the CTRL–S character as a command to disable roll mode.

A VT100 terminal in AUTO–XOFF mode *does* recognize CTRL–S and CTRL–Q as flow control characters; so does a VT220 terminal when it is not in its NO XOFF mode. VT100 support suppresses output of these two characters to prevent applications from accidentally hanging the terminal by sending D200 terminal commands such as disable roll mode.

Entering the ESC Character

A VT100 terminal uses the ESC character to start a command sequence. When VT100 support sees an ESC character, it interprets this character in the context of the characters that follow it. This means that an ESC character cannot stand alone.

Since many Data General applications treat ESC as valid data, you must be able to enter this character on a VT100 terminal. To allow this, VT100 support treats two consecutive ESC characters as one ESC character that you are entering as data.

Note the following characteristics of this two–keystroke scheme:

- The first ESC does not echo. The second ESC produces the same result as typing one ESC on a DASHER D200 terminal. Depending on the application the second ESC can echo as a $, cause a console interrupt, or perform some application–specific action.

- If the first ESC is followed by an unrecognizable character (for example, the DEL character), the system will ignore both characters and ring the terminal's bell.

- You can enter a command sequence by pressing the ESC key and typing standard alphanumeric and punctuation characters. For example, if you press ESC and type [and A:

  ESC [A

  they will not echo and the system will treat this sequence the same way as it would if you pressed the uparrow key.

- If you press the ESC key once and follow it with a VT100 key that initiates a command sequence, the system will first see two consecutive ESC characters. They are the ESC key you pressed and the first character of the command sequence. AOS/VS and AOS/VS II treat the pair as a single ESC entered as data. Next, the system treats the rest of the command sequence as data.

If your terminal is a VT220, it does not have an ESC key. To learn how to simulate ESC, see the next section.

Data Entry Errors

The VT100 support software recognizes the start of a character sequence that needs translating. In fact, it recognizes such a sequence by the first character.

If the sequence doesn't end in a recognizable way, then the translation scheme ignores the characters. In the case of input it also rings the bell and ignores any other typed–ahead input (it flushes the input ring buffer).

If you start to type in a multi–character sequence for ESC or a function key, and you realize you made a mistake, then you can abort the sequence by typing any character that makes the sequence unrecognizable. Try the Delete or Return key. The bell will tell you that the software rejected the characters.

Character Attributes

DASHER D200 and VT100 terminals support the same basic character attributes. VT100 translation supports VT100 attributes so that they behave the same way as D200 attributes — with one difference.

Both D200 and VT100 terminals have commands that allow you to control the blinking of individual characters and groups of characters wherever they appear on a screen. D200 terminals have a command, disable blink, that allows you to override the individual blink attributes of characters on the screen so that they no longer blink. VT100 terminals do not have such a global override command for blinking. So, VT100 support suppresses output of this command.

In general, don't mix VT100 and D200 command sequences to change character attributes such as blinking. Use one command sequence or the other, preferably the D200 one. If you mix them the results are undefined.

Simulating DASHER D200 Function Keys

A DASHER D200 terminal has 15 function keys. They generate different character sequences depending on whether the key is pressed alone or in combination with the SHIFT key, the CTRL key, or both.

A VT100 terminal has only four function keys while a VT220 terminal has 15 function keys. Both terminals generate the same character sequences from their function keys regardless of whether the SHIFT or CTRL keys are also held down.

VT100 support allows you to simulate a DASHER D200 function key. In general, you do this by typing a multiple character sequence.

VT100 support provides four schemes for you to enter the equivalent of D200 function keys. All four schemes use function keys. As a VT220 user you can employ all four schemes while as a VT100 user you can employ only the first two schemes.

Three of these four schemes require multiple keystrokes.

Function Key Scheme 1 — VT100 or VT220 Terminals

This scheme lets you simulate up to four function keys. You must use two keys. Suppose you have a VT100 or VT220 terminal and want to enter the equivalent of D200 function keys 1, 2, 3, and 4 without the SHIFT and CTRL keys. Use the following list.

To simulate D200 key	Use VT100/VT220 keys
F1	PF1 PF1 (Press PF1 twice.)
F2	PF2 PF2
F3	PF3 PF3
F4	PF4 PF4

For example, assume you are using the SED text editor on a VT100 terminal and want to move the cursor down one screen. The function key that does this is F2 (on the SED function key template, it is labeled DOWN ONE SCREEN). So from the F2 entry in the table you can see that you would press PF2 twice.

Function Key Scheme 2 — VT100 or VT220 Terminals

This scheme lets you simulate up to 76 function keys. You must use three keys. Suppose you have a VT100 or VT220 terminal and want to enter the equivalent of any D200 function key, including those that also require the SHIFT and CTRL keys. Use the following list where n is a one– or two–digit number between 1 and 15. If n is less than 10, you must insert a leading zero.

The symbol Ci in the left column represents the D200 keys C1, C2, C3, and C4. These keys are in the cursor control keypad to the right of the main keypad. No Data General program supplied with the operating system uses the Ci keys, but one of your applications may do so.

To simulate D200 key	Use VT100/VT220 keys	Example
Fn	PF1 n n	For F1, use PF1 0 1.
SHIFT Fn	PF2 n n	For SHIFT–F1, use PF2 0 1.
CTRL Fn	PF3 n n	For CTRL–F1, use PF3 0 1.
CTRL/SHIFT Fn	PF4 n n	For CTRL–SHIFT–F1, use PF4 0 1.
Ci	PF1 i+15	For C1, use PF1 1 6.
SHIFT Ci	PF2 i+15	For SHIFT–C1, use PF2 1 6.

For example, assume you are using CEO on a VT100 terminal and want to delete a word. The function key that does this is SHIFT F9 (on the CEO template for D200 terminals, the key is labeled DELETE WORD). You would press the following three keys.

```
PF2 0 9
```

Function Key Scheme 3 — VT220 Terminals

This scheme lets you simulate up to 15 function keys using just one key. Suppose you have a VT220 terminal and want to simulate D200 function keys 1 through 15 without the SHIFT and CTRL keys. Use the following list.

To simulate D200 key	Use VT220 key
F1	F6
F2	F7
F3	F8
Fn	F(n+5)
F15	F20

For example, assume you are using the SED text editor on a VT220 terminal and want to move the cursor down one screen. The function key that does this is F2 (on the SED function key template, it is labeled DOWN ONE SCREEN). So from the F2 entry in the table you can see that you would press PF7 (adding 5 to the number of the D200 function key).

Function Key Scheme 4 — VT220 Terminals

This scheme lets you simulate up to 76 function keys. For 60 of these, you can use just two keys. Suppose you have a VT220 terminal and want to enter the equivalent of any D200 function key, including those that also require the SHIFT and CTRL keys. Use the following list where n is a one– or two–digit number between 1 and 15. Use the following list where n in the first four rows of the left column is a one– or two–digit number between 1 and 15. The symbol Cn in the left column represents the D200 function keys C1, C2, C3, and C4. The symbols in the right column represent the pressing of two different function keys (for the first four rows) or one function key and two different nonfunction keys (for the last two rows).

The symbol Ci in the left column represents the D200 keys C1, C2, C3, and C4. These keys are in the cursor control keypad to the right of the main keypad. No Data General program supplied with the operating system uses the Ci keys, but one of your applications may do so.

To simulate D200 key	Use VT220 keys	Example
Fn	PF1 F(n+5)	For F1, use PF1 F6.
SHIFT Fn	PF2 F(n+5)	For SHIFT–F1, use PF2 F6.
CTRL Fn	PF3 F(n+5)	For CTRL–F1, use PF3 F6.
CTRL/SHIFT Fn	PF4 F(n+5)	For CTRL–SHIFT–F1, use PF4 F6.
Ci	PF1 i+15	For C1, use PF1 1 6 .
SHIFT Ci	PF2 i+15	For SHIFT–C1, use PF2 1 6.

For example, assume you are using CEO on a VT100 terminal and want to delete a word. The function key for this is SHIFT F9 (on the CEO template DG terminals, the key is labeled DELETE WORD). You would press the following two keys.

PF2 F14

Example Session

Let's look over the shoulder of user Jeff as he uses his VT220 terminal. Jeff's terminal is turned on and he sees a message such as

**** *AOS/VS Rev x.xx.xx.xx / Press NEW LINE to begin logging on* **** #

In this banner the number sign (#) represents the VT220 cursor. The cursor may actually appear as a blinking vertical box.

Jeff presses the key RETURN and logs on. Each time he presses RETURN, the cursor moves down only, not down and to the left as it would on a Data General terminal. Finally a message and CLI prompt appear:

AOS/VS CLI32 Rev xx.xx.xx.xx 24–Feb–91 10:57:14
)

Jeff wants his VT220 terminal to behave as closely as possible to a DASHER 200 terminal, and he wants to use software that accepts the ESC character. He gives the CLI command

```
) CHARACTERISTICS/XLT/OTT ↵
```

If his terminal were a VT100 he wouldn't need the /OTT switch.

Now Jeff can use his VT220 terminal to emulate a DASHER D200 terminal. In particular, the screenedit keys work the same way on both terminals so, for example, that Jeff can use CTRL–A to redisplay the previous CLI command.

Jeff decides to edit the last page of file MYPROG.F77 by using the SED text editor.

```
) XEQ SED MYPROG.F77 ↵
```

SED Rev x.xx.xx.xx; Input file – :UDD1:JEFF:MYPROG.F77
view
*
page 2 line 40

Jeff wants to go to the middle line of the last page and modify it. He wants to use the function key SHIFT F5 to do this. Jeff reviews function key scheme 4. The list says he should press the two function keys PF2 and F(5+n). In this case, n equals 5 (from F5). So he presses the keys PF2 F10. (If Jeff were using a VT100 terminal, he would review function key scheme 2 and press the three keys PF2 0 5.)

Jeff sees the first lines of the last page. His next step is to go to the middle of the page. No SED command does this directly, but the SED template says that CTRL F3 will perform the function MIDDLE LINE ON PAGE. Jeff reviews scheme 4. The list says he should press the two function keys PF3 and F(5+n); here, n is 3 from F3 so he presses the keys PF3 F8. (If Jeff were using a VT100 terminal, he would review scheme 2 and press the three keys PF3 0 3.)

Jeff sees the lines centered around the middle line. To modify the middle line he can use the SED command MODIFY. Instead, he looks at the SED template to learn that function key F6 will also give this command. Scheme 4 suggests the two function keys PF1 and F(5+n) or PF1 F11. Or, scheme 3 suggest the one function key F(5+n) which is F11. To exit from MODIFY mode with the ESC character, he recalls the earlier section "Entering the ESC Character" and presses CTRL–[, or the tilde key (~) or the right brace key (}) twice. The second key pressed sends the modification. Jeff can use the familiar screenedit keys while SED executes its MODIFY command. One of them is CTRL–F to move the cursor forward one word. (If Jeff were using a VT100 terminal, he would review scheme 2 and press the three keys PF1 0 6. He would enter the ESC character by pressing the ESC key twice regardless of the setting of the /OTT switch. Or, if he had set the /OTT switch he could press the tilde key (~) or the right brace key (}) twice.)

Jeff has finished editing his file. He leaves the SED editor and then logs off the operating system. The system logs him off and the VT220 terminal awaits its next user.

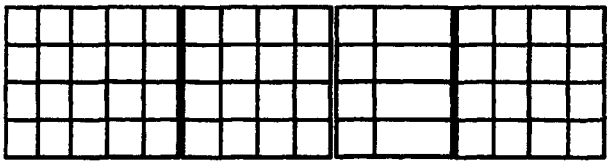
Using a Template

You might find it useful to make a template for your VT100 or VT220 terminal if your Data General application program makes extensive use of function keys. You can make it from heavyweight paper and place it near your VT100 or VT220's function keys. Once the application is running, you can use the section "Entering DASHER D200 Function Keys," to decide what VT100 or VT220 keys to press.

For example, suppose your application is Data General's CEO office automation software and your terminal is a VT220. Begin by noting that the 15 VT220 function keys you need fall into four classes because of their physical grouping and location:

F6 F7 F8 F9 F10
F11 F12 F13 F14
F15 F16
F17 F18 F19 F20

You will find it helpful to look at a VT220 keyboard or a picture of it and at a DASHER D200 template for CEO as you read the rest of this section. You must measure the VT220 keyboard as you draw the template. In particular, a VT220 function key 16 is much wider than the other function keys. The resulting template appears as follows:



Sections of the template appear next with labeled keys, in four parts. The template you would make contains these four parts from left to right. Each of the 15 columns in the template is directly above a VT220 function key. The first of these 15 columns is directly above VT220 function key F6.

Part 1 of 4 follows. It is the leftmost part of your template.

	F6 (DG F1)	F7 (DG F2)	F8 (DG F3)	F9 (DG F4)	F10 (DG F5)
CTRL/ SHIFT					
CTRL	SWITCH WINDOWS	COLUMN	MERGE	END PAGE	VIEW/EDIT
SHIFT	HELP	INDEX	FORMAT RULER	BEGIN/END LINE	GO TO
	EXECUTE	COMMAND	PREVIOUS SCREEN	NEXT SCREEN	INTERRUPT

Part 2 of 4 follows. It is in the left half of your template.

F11 (DG F6)	F12 (DG F7)	F13 (DG F8)	F14 (DG F9)
REPLACE	MOVE	COPY	INSERT BLOCK
FIND	TEXT ATTRIBUTE	CENTER	DELETE WORD
INSERT	INSERT SPACE	DELETE CHAR	DELETE

Part 3 of 4 follows. It is in the right half of your template.

F15 (DG F10)	F16 (DG F11)
GLOBAL REPLACE	FOOTNOTE
SPELL	BACK FIELD
INDENT	CANCEL/EXIT

Part 4 of 4 follows. It is the rightmost part of your template.

F17 (DG F12)	F18 (DG F13)	F19 (DG F14)	F20 (DG F15)
BLOCK PROTECT	PLACEMARK	OVERLAY BLOCK	FORMAT CELLS
INCLUDE	DISCR. HYPHEN	ERASE	ABSOLUTE/ RELATIVE
READ	SAVE	CALCU- LATOR	DEFINE RANGE

For example, suppose your application is CEO and you want to find the next occurrence of the string *sentance* in the current document so that you can correct its spelling. The CEO template tells you that the FIND key is F6. Function key scheme 4 tells you to press the two function keys PF2 F(5+n), which is PF2 F(5+6), which is PF2 F11.

Notice that VT220 function key F11 is directly under the box labeled FIND on the template; the template is designed and placed on the VT220 keyboard with this convenient columnar alignment in mind.

Finally, a generalization of this example leads to a possible modification of the left part of the template. The top row of keys in function key scheme 4 correspond to the labels of the four rows of the template (in inverse order). If you use scheme 4, you can change the template so that its left part begins as follows.

	F6 (DG F1)	F7 (DG F2)
CTRL/SHIFT= PF4		
CTRL = PF3	GLOBAL REPLACE	FOOTNOTE
SHIFT = PF2	SPELL	BACK FIELD
PF1	INDENT	CANCEL/EXIT

When you use this template on a VT220 terminal, you need not continually reread the list under function key scheme 4. Just find the CEO command you need on the template and look to its left and above it to find the two function keys to press.

For instance, recall the previous need to enter the FIND command. To the left of the block labeled FIND is PF2 and above this block is F11. So, the two keys to press are PF2 F11. For another example, suppose you need to give the PLACEMARK command to CEO. To the left of the block that contains PLACEMARK is PF3 and above this block is F18. So the two keys to press are PF3 and F18.

End of Appendix

Index

Symbols

Numbers

A

B

C

D

E

F

G

H

J

K

L

M

N

O

P

086-000200 updates
093-000646

Q

R

S

T

U

V

W

X

V

W

X

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.

TO ORDER

1. An order can be placed with the TIPS group in two ways:
   a) MAIL ORDER – Use the order form on the opposite page and fill in all requested information. Be sure to include shipping charges and local sales tax. If applicable, write in your tax exempt number in the space provided on the order form.

      Send your order form with payment to:
      Data General Corporation
      ATTN: Educational Services/TIPS G155
      4400 Computer Drive
      Westboro, MA 01581–9973

   b) TELEPHONE – Call TIPS at (508) 870–1600 for all orders that will be charged by credit card or paid for by purchase orders over $50.00. Operators are available from 8:30 AM to 5:00 PM EST.

METHOD OF PAYMENT

2. As a customer, you have several payment options:
   a) Purchase Order – Minimum of $50. If ordering by mail, a hard copy of the purchase order must accompany order.
   b) Check or Money Order – Make payable to Data General Corporation.
   c) Credit Card – A minimum order of $20 is required for MasterCard or Visa orders.

SHIPPING

3. To determine the charge for UPS shipping and handling, check the total quantity of units in your order and refer to the following chart:

Total Quantity	Shipping & Handling Charge
1–4 Items	$5.00
5–10 Items	$8.00
11–40 Items	$10.00
41–200 Items	$30.00
Over 200 Items	$100.00

   If overnight or second day shipment is desired, this information should be indicated on the order form. A separate charge will be determined at time of shipment and added to your bill.

VOLUME DISCOUNTS

4. The TIPS discount schedule is based upon the total value of the order.

Order Amount	Discount
$0–$149.99	0%
$150–$499.99	10%
Over $500	20%

TERMS AND CONDITIONS

5. Read the TIPS terms and conditions on the reverse side of the order form carefully. These must be adhered to at all times.

DELIVERY

6. Allow at least two weeks for delivery.

RETURNS

7. Items ordered through the TIPS catalog may not be returned for credit.
8. Order discrepancies must be reported within 15 days of shipment date. Contact your TIPS Administrator at (508) 870–1600 to notify the TIPS department of any problems.

INTERNATIONAL ORDERS

9. Customers outside of the United States must obtain documentation from their local Data General Subsidiary or Representative. Any TIPS orders received by Data General U.S. Headquarters will be forwarded to the appropriate DG Subsidiary or Representative for processing.

TIPS ORDER FORM

Mail To: Data General Corporation
Attn: Educational Services/TIPS G155
4400 Computer Drive
Westboro, MA 01581 - 9973

BILL TO:	SHIP TO: (No P.O. Boxes. Complete Only If Different Address)
COMPANY NAME _____	COMPANY NAME _____
ATTN: _____	ATTN: _____
ADDRESS _____	ADDRESS (NO PO BOXES) _____
CITY _____	CITY _____
STATE _____ ZIP _____	STATE _____ ZIP _____

Priority Code _____ (See label on back of catalog)

Authorized Signature of Buyer (Agrees to terms & conditions on reverse side)	Title	Date	Phone (Area Code)	Ext.

ORDER #	QTY	DESCRIPTION	UNIT PRICE	TOTAL PRICE

A SHIPPING & HANDLING

☐ UPS	ADD
1–4 Items	$5.00
5–10 Items	$8.00
11–40 Items	$10.00
41–200 Items	$30.00
200+ Items	$100.00

Check for faster delivery

Additional charge to be determined at time of shipment and added to your bill.

☐ UPS Blue Label (2 day shipping)
☐ Red Label (overnight shipping)

B VOLUME DISCOUNTS

Order Amount	Save
$0–$149.99	0%
$150–$499.99	10%
Over $500.00	20%

Tax Exempt # or Sales Tax (if applicable) _____

ORDER TOTAL	
Less Discount See B	–
SUB TOTAL	
Your local* sales tax	+
Shipping and handling – See A	+
TOTAL – See C	

C PAYMENT METHOD

☐ Purchase Order Attached ($50 minimum)
P.O. number is _____. (Include hardcopy P.O.)
☐ Check or Money Order Enclosed
☐ Visa ☐ MasterCard ($20 minimum on credit cards)

Account Number _____ Expiration Date _____

Authorized Signature _____

(Credit card orders without signature and expiration date cannot be processed.)

THANK YOU FOR YOUR ORDER

PRICES SUBJECT TO CHANGE WITHOUT PRIOR NOTICE.
PLEASE ALLOW 2 WEEKS FOR DELIVERY.
NO REFUNDS NO RETURNS.

* Data General is required by law to collect applicable sales or use tax on all purchases shipped to states where DG maintains a place of business, which covers all 50 states. Please include your local taxes when determining the total value of your order. If you are uncertain about the correct tax amount, please call 508–870–1600.

Form 702
Rev. 8/87

DATA GENERAL CORPORATION
TECHNICAL INFORMATION AND PUBLICATIONS SERVICE
TERMS AND CONDITIONS

Data General Corporation ("DGC") provides its Technical Information and Publications Service (TIPS) solely in accordance with the following terms and conditions and more specifically to the Customer signing the Educational Services TIPS Order Form. These terms and conditions apply to all orders, telephone, telex, or mail. By accepting these products the Customer accepts and agrees to be bound by these terms and conditions.

1. CUSTOMER CERTIFICATION
Customer hereby certifies that it is the owner or lessee of the DGC equipment and/or licensee/sub-licensee of the software which is the subject matter of the publication(s) ordered hereunder.

2. TAXES
Customer shall be responsible for all taxes, including taxes paid or payable by DGC for products or services supplied under this Agreement, exclusive of taxes based on DGC's net income, unless Customer provides written proof of exemption.

3. DATA AND PROPRIETARY RIGHTS
Portions of the publications and materials supplied under this Agreement are proprietary and will be so marked. Customer shall abide by such markings. DGC retains for itself exclusively all proprietary rights (including manufacturing rights) in and to all designs, engineering details and other data pertaining to the products described in such publication. Licensed software materials are provided pursuant to the terms and conditions of the Program License Agreement (PLA) between the Customer and DGC and such PLA is made a part of and incorporated into this Agreement by reference. A copyright notice on any data by itself does not constitute or evidence a publication or public disclosure.

4. LIMITED MEDIA WARRANTY
DGC warrants the CLI Macros media, provided by DGC to the Customer under this Agreement, against physical defects for a period of ninety (90) days from the date of shipment by DGC. DGC will replace defective media at no charge to you, provided it is returned postage prepaid to DGC within the ninety (90) day warranty period. This shall be your exclusive remedy and DGC's sole obligation and liability for defective media. This limited media warranty does not apply if the media has been damaged by accident, abuse or misuse.

5. DISCLAIMER OF WARRANTY
EXCEPT FOR THE LIMITED MEDIA WARRANTY NOTED ABOVE, DGC MAKES NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRANTIES OF MERCHANTABILITY AND FITNESS FOR PARTICULAR PURPOSE ON ANY OF THE PUBLICATIONS, CLI MACROS OR MATERIALS SUPPLIED HEREUNDER.

6. LIMITATION OF LIABILITY
A. CUSTOMER AGREES THAT DGC'S LIABILITY, IF ANY, FOR DAMAGES, INCLUDING BUT NOT LIMITED TO LIABILITY ARISING OUT OF CONTRACT, NEGLIGENCE, STRICT LIABILITY IN TORT OR WARRANTY SHALL NOT EXCEED THE CHARGES PAID BY CUSTOMER FOR THE PARTICULAR PUBLICATION OR CLI MACRO INVOLVED. THIS LIMITATION OF LIABILITY SHALL NOT APPLY TO CLAIMS FOR PERSONAL INJURY CAUSED SOLELY BY DGC'S NEGLIGENCE. OTHER THAN THE CHARGES REFERENCED HEREIN, IN NO EVENT SHALL DGC BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL OR CONSEQUENTIAL DAMAGES WHATSOEVER, INCLUDING BUT NOT LIMITED TO LOST PROFITS AND DAMAGES RESULTING FROM LOSS OF USE, OR LOST DATA, OR DELIVERY DELAYS, EVEN IF DGC HAS BEEN ADVISED, KNEW OR SHOULD HAVE KNOWN OF THE POSSIBILITY THEREOF; OR FOR ANY CLAIM BY ANY THIRD PARTY.

B. ANY ACTION AGAINST DGC MUST BE COMMENCED WITHIN ONE (1) YEAR AFTER THE CAUSE OF ACTION ACCRUES.

7. GENERAL
A valid contract binding upon DGC will come into being only at the time of DGC's acceptance of the referenced Educational Services Order Form. Such contract is governed by the laws of the Commonwealth of Massachusetts, excluding its conflict of law rules. Such contract is not assignable. These terms and conditions constitute the entire agreement between the parties with respect to the subject matter hereof and supersedes all prior oral or written communications, agreements and understandings. These terms and conditions shall prevail notwithstanding any different, conflicting or additional terms and conditions which may appear on any order submitted by Customer. DGC hereby rejects all such different, conflicting, or additional terms.

8. IMPORTANT NOTICE REGARDING AOS/VS INTERNALS SERIES (ORDER #1865 & #1875)
Customer understands that information and material presented in the AOS/VS Internals Series documents may be specific to a particular revision of the product. Consequently user programs or systems based on this information and material may be revision-locked and may not function properly with prior or future revisions of the product. Therefore, Data General makes no representations as to the utility of this information and material beyond the current revision level which is the subject of the manual. Any use thereof by you or your company is at your own risk. Data General disclaims any liability arising from any such use and I and my company (Customer) hold Data General completely harmless therefrom.

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

093–000646–01

Cut here and insert in binder spine pocket