VMS

digital

VMS DCL Concepts Manual

Order Number AA–LA10A–TE

VMS DCL Concepts Manual

Order Number: AA–LA10A–TE

April 1988

This manual describes the VMS DIGITAL Command Language (DCL).

Revision/Update Information:	This manual supersedes the *VAX/VMS DCL Concepts Manual,* Version 4.4.
Software Version:	VMS Version 5.0

digital equipment corporation
maynard, massachusetts

April 1988

The postpaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DEC	DIBOL	UNIBUS
DEC/CMS	EduSystem	VAX
DEC/MMS	IAS	VAXcluster
DECnet	MASSBUS	VMS
DECsystem–10	PDP	VT
DECSYSTEM–20	PDT	
DECUS	RSTS	
DECwriter	RSX	digital™

ZK4496

HOW TO ORDER ADDITIONAL DOCUMENTATION
DIRECT MAIL ORDERS

USA & PUERTO RICO*

Digital Equipment Corporation
P.O. Box CS2008
Nashua, New Hampshire
03061

CANADA

Digital Equipment
of Canada Ltd.
100 Herzberg Road
Kanata, Ontario K2K 2A6
Attn: Direct Order Desk

INTERNATIONAL

Digital Equipment Corporation
PSG Business Manager
c/o Digital's local subsidiary
or approved distributor

In Continental USA and Puerto Rico call 800-258-1710.

In New Hampshire, Alaska, and Hawaii call 603-884-6660.

In Canada call 800-267-6215.

*Any prepaid order from Puerto Rico must be placed with the local Digital subsidiary (809-754-7575).

Internal orders should be placed through the Software Distribution Center (SDC), Digital Equipment Corporation, Westminster, Massachusetts 01473.

Production Note

This book was produced with the VAX DOCUMENT electronic publishing system, a software tool developed and sold by DIGITAL. In this system, writers use an ASCII text editor to create source files containing text and English-like code; this code labels the structural elements of the document, such as chapters, paragraphs, and tables. The VAX DOCUMENT software, which runs on the VMS operating system, interprets the code to format the text, generate a table of contents and index, and paginate the entire document. Writers can print the document on the terminal or line printer, or they can use DIGITAL-supported devices, such as the LN03 laser printer and PostScript™ printers (PrintServer 40 or LN03R ScriptPrinter), to produce a typeset-quality copy containing integrated graphics.

™ PostScript is a trademark of Adobe Systems, Inc.

Contents

TABLES

Preface

The *VMS DCL Concepts Manual* provides an overview of the DIGITAL Command Language. Read it to learn the syntax for using DCL commands. It also covers symbols and logical names, two structures that let you tailor DCL to suit your needs. Before using this manual you should read the *Introduction to VMS*.

Intended Audience

This manual is intended for anyone who uses the VMS operating system.

Document Structure

The *VMS DCL Concepts Manual* is organized into the following eight chapters.

- **Chapter 1, The DCL Command Line**

  Describes the syntax rules for entering DCL commands. It explains how to enter parameters, qualifiers, and comments, as well as date and time values.

- **Chapter 2, Editing the DCL Command Line**

  Describes how to execute, recall, and cancel DCL commands. It explains how to edit the command line.

- **Chapter 3, File Specifications**

  Describes the syntax rules for entering node, device, directory, and file specifications. It tells you how the system interprets the components of a file specification, including the default value that is assumed if you omit anything. It also shows you how to use wildcard characters to indicate groups of files.

- **Chapter 4, Logical Names**

  Explains how to use logical names to refer to devices, directories, and files.

- **Chapter 5, Symbols**

  Shows you how to create and use symbols. Symbols have a variety of uses such as providing customized abbreviations for commands and passing variables in command procedures.

- **Chapter 6, More on Expressions**

  Describes the rules for creating character string and numeric expressions. It discusses how DCL interprets the different operations and comparisons you can perform.

- **Chapter 7, Symbol Substitution**

  Explains how the DCL command interpreter replaces symbol names with their current values when it processes a command.

- **Chapter 8, UIC-Based Protection**

  Explains how to protect devices, directories, and files with user identification codes (UICs).

Associated Documents

Overview of VMS Documentation—describes the new organization of the document set.

Introduction to VMS—provides an introduction to the VMS operating system and the DIGITAL Command Language. This manual is especially recommended for novice users.

VMS DCL Dictionary—contains a detailed description of each DCL command and lexical function.

Guide to Using VMS Command Procedures— explains how to write command procedures with DCL commands and lexical functions.

VMS System Messages and Recovery Procedures Reference Manual—explains error and warning messages you might receive.

Conventions

Convention	Meaning
RET	in Examples, a key name (usually abbreviated) shown within a box indicates that you press a key on the keyboard; in text,a key name is not enclosed in a box. In this example, the key is the RETURN key. (Note that the RETURN key is not usually shown in syntax statements or in all examples; however, assume that you must press the RETURN key after entering a command or responding to a prompt.)
CTRL/C	A key combination, shown in uppercase with a slash separating two key names, indicates that you hold down the first key while you press the second key. For example, the key combination CTRL/C indicates that you hold down the key labeled CTRL while you press the key labeled C. In examples, a key combination is enclosed in a box.
$ SHOW TIME 05-JUN-1988 11:55:22	In examples, system output (what the system displays) is shown in black. User input (what you enter) is shown in red.
$ TYPE MYFILE.DAT . . .	In examples, a vertical series of periods, or ellipsis, means either that not all the data that the system would display in response to a command is shown or that not all the data a user would enter is shown.

Convention	Meaning
input-file, . . .	In examples, a horizontal ellipsis indicates that additional parameters, values, or other information can be entered, that preceding items can be repeated one or more times, or that optional arguments in a statement have been omitted.
[logical-name]	Brackets indicate that the enclosed item is optional. (Brackets are not, however, optional in the syntax of a directory name in a file specification or in the syntax of a substring specification in an assignment statement.)
quotation marks apostrophes	The term quotation marks is used to refer to double quotation marks ("). The term apostrophe (') is used to refer to a single quotation mark.

1 The DCL Command Line

DIGITAL Command Language (DCL) is the language you use to communicate with the VMS operating system. DCL commands let you do the following tasks:

- Get information about the system
- Work with files
- Work with disks, magnetic tapes, and other devices
- Modify your work environment
- Develop and execute programs
- Provide security and ensure that resources are used efficiently

This chapter explains the format for entering DCL commands. It describes the parts of a command line and the rules for the following:

- Entering commands
- Entering parameters
- Entering qualifiers
- Entering date and time values

1.1 Entering Commands

You can use DCL in the following two modes:

- Interactive. In interactive mode, you enter commands from your terminal. A command has to finish executing before you can enter another one.
- Noninteractive. In noninteractive mode, the system creates another process to execute commands on your behalf. Batch jobs and network processes use DCL in noninteractive mode. For example, when you submit a command procedure as a batch job, a separate process is created to execute the command procedure. You can continue to use your terminal interactively while the batch job runs.

DCL commands consist of words from the English language (generally verbs) that describe what you want the system to do. When you type a command and press the RETURN key, it is read and translated by the DCL command interpreter. The way the command interpreter responds to a command is determined by the type of command entered. The three types of commands are as follows:

- Built-in commands. These commands are built into the command interpreter. DCL executes a built-in command directly. The DCL built-in commands are listed in Table 1–1.

- Commands that call command images. DCL calls another program to execute the command. A program that is called to execute a command is referred to as a *command image*. Parameter and qualifier information are passed to the program. Most commands not listed in Table 1–1 are in this category.

  A command image can be VMS- or user-supplied. All VMS-supplied DCL commands are described in the *VMS DCL Dictionary*.

- Commands that call interactive programs. DCL calls the program associated with the command. You then work interactively with the program until you exit from it and return to DCL command level. For example, the MAIL command calls an interactive program.

Table 1–1 Built-in Commands

=,==,:=,:==	ALLOCATE	ASSIGN
ATTACH	CALL	CANCEL
CLOSE	CONNECT	CONTINUE
CREATE/NAME_TABLE	DEALLOCATE	DEASSIGN
DEBUG	DECK	DEFINE
DEFINE/KEY	DELETE/KEY	DELETE/SYMBOL
DEPOSIT	DISCONNECT	ENDSUBROUTINE
EOD	EXAMINE	EXIT
GOSUB	GOTO	IF
INQUIRE	ON	OPEN
READ	RECALL	RETURN
SET CONTROL	SET DEFAULT	SET KEY
SET ON	SET OUTPUT_RATE	SET PROMPT
SET PROTECTION /DEFAULT	SET UIC	SET VERIFY
SHOW DEFAULT	SHOW KEY	SHOW QUOTA
SHOW PROTECTION	SHOW STATUS	SHOW SYMBOL
SHOW TIME	SHOW TRANSLATION	SPAWN
STOP	SUBROUTINE	WAIT
WRITE		

1.1.1 The Parts of a Command Line

The DCL command line can contain the following components:

$	The dollar sign ($) is the DCL prompt. When you work interactively, DCL displays the prompt when it is ready to accept a command. When you write a command procedure, you must type the dollar sign ($) at the beginning of each line.
Label	Identifies a line in a command procedure. Use labels only within command procedures.
Command	Specifies the name of the command.
Parameter	Specifies what the command acts upon. Examples of parameters are file specifications, queue names, and logical names.
Qualifier	Modifies the action taken by the command. Some qualifiers can modify parameters. Some can accept values.
Value	Modifies a qualifier. A value can be a filename, a character string, a number, or a DCL *keyword.* A keyword is a word that has special meaning in DCL. For example, GROUP, WORLD, and OWNER are DCL keywords. A DCL keyword may also have a value.

The way in which the parts of a command line are put together is referred to as the *command line syntax.* DCL command line syntax follows the general format shown below (items in square brackets [] are optional).

```
[$] [label:]  command  [/qualifier[=value]...] [parameter[/qualifier...]]
```

Observe the following rules when entering DCL commands:

- You can use any combination of uppercase and lowercase letters. The DCL command interpreter translates lowercase letters to uppercase.
- A DCL command line can contain up to 1024 characters after symbols and lexical functions have been expanded. There can be 256 characters on a line. For information on extending a command over multiple lines, see Section 1.1.3.
- A command line can contain a maximum of 128 elements (for example, a file specification or qualifier). Each element can have up to 255 characters.
- At least one blank space must separate the command name from the first parameter, and at least one blank must separate each additional parameter from the previous parameter. Multiple spaces and tabs are permitted in all cases where a single blank is required. The command interpreter compresses multiple blank spaces or tabs to a single blank space.
- Each qualifier must be preceded with a slash (/). The slash can be preceded or followed by any number of spaces or tabs.
- A label must be terminated with a colon. The colon can be preceded or followed by any number of spaces or tabs. A label name can have up to 255 characters, but it cannot include blank spaces.

The DCL Command Line

1.1 Entering Commands

The following is an example of a DCL command line:

```
$ PRINT/COPIES=10 FOOD.LST
```

where:

- *$* is the DCL prompt
- *PRINT* is a command
- */COPIES* is a qualifier that modifies the command
- *10* is a value that modifies the /COPIES qualifier
- *FOOD.LST* is a parameter (in this case the parameter is a file specification)

1.1.2 Command Prompting

If you enter a command that requires parameters and you do not specify the parameters, the DCL command interpreter asks you for them. For example:

```
$ PRINT/COPIES=15
_File:
```

The PRINT command expects a file specification. If you do not enter one, the system asks you for it. A line beginning with an underscore (_) means that the system is waiting for your response.

On any prompt, you can enter one or more of the remaining parameters and any additional qualifiers. When you are asked for an optional parameter, you can press RETURN to omit it.

If you press CTRL/Z after a command prompt, DCL ignores the command and redisplays the DCL prompt.

1.1.3 Continuing Commands on More Than One Line

If you want to type a lengthy command line that includes many qualifiers, you can extend it over multiple lines to make it more readable.

Extend a command line by typing a hyphen (-), pressing RETURN, and typing the rest of the command on the next line. The DCL command interpreter responds with an underscore followed by a dollar sign (_$). This indicates that DCL is still accepting the command. For example:

```
$ TYPE -
_$ TAXES.MEM
```

Follow these rules for continuing a command line:

- Enter the hyphen as the last element on the line or the last element before a comment. A hyphen not used at the end of a line is treated like any other character and is valid in file specifications.
- There is no restriction on the number of continued lines you can use to enter a command, as long as you do not exceed the 1024-character limit.
- The RETURN after a hyphen is not treated as a delimiter. You must supply any required spaces.

Command line continuation is also used in command procedures when you want to make a procedure more readable. In command procedures, do not begin continuation lines with dollar signs. For example:

```
$  PRINT LAB.DAT -
 /AFTER=17:00 -
 /COPIES=20 -
 /NAME="COMGUIDE"
```

For more information on command procedures, see the *Guide to Using VMS Command Procedures.*

1.1.4 Entering Comments

Indicate a comment by preceding it with an exclamation point (!). DCL considers everything to the right of an exclamation point to be a comment. It ignores this information when processing the command line. Comments are valid in the following positions:

- As the first item in a command line. In this case, the entire line is considered a comment and is not processed.
- After the last character in a command line or after a hyphen continuation character.

Some examples follow:

```
$ !THIS ENTIRE LINE IS A COMMENT
$ PRINT LAB.DAT-  !  PRINT COMMAND COMMENT
_$ /COPIES=3    !  3 COPIES, PLEASE
```

1.1.5 Abbreviating Command Names

When the DCL command interpreter examines a command line, it looks only at the first four characters of a command. All command names are unique within four characters. This means that you can abbreviate any command by typing only the first four characters.

You can abbreviate a command name to fewer than four characters as long as the abbreviated name remains unique among all DCL command names. For example, the HELP command is currently the only command that begins with the letter "H." Therefore, the HELP command can be truncated to just one character. The DEALLOCATE and DEASSIGN commands, however, have the same first three characters, so these commands need at least the first four characters to be unique.

The following commands are exceptions to the minimum truncation rule because they can be truncated to their first character, even though other commands begin with the same character:

CONTINUE
DEPOSIT
EXAMINE
RUN

You can also abbreviate qualifiers and keywords. For more information, see Section 1.3.5.

1.1.6 Abbreviations in Command Procedure Files

You should not abbreviate commands, qualifiers, or keywords in command procedures for the following reasons:

- Your command procedures will be difficult to read.
- The abbreviations may not be valid after new DCL commands are added.

1.2 Entering Parameters

The parameter section of the command descriptions in the *VMS DCL Dictionary* shows the parameters that are accepted by each command. The following rules apply:

- Square brackets [] indicate optional items. For example:

  ```
  SHOW QUEUE [queue-name]
  ```

  The square brackets mean that entering a queue name is optional.

  Anything not enclosed in square brackets is required. For example:

  ```
  SET HOST node-name
  ```

  You must enter a node name with the SET HOST command.

- Required parameters must be placed to the left of optional parameters.
- An input file parameter must precede an output file parameter.
- A parameter may be one item or a series of items. If you enter a series of items, separate them with commas (,) or plus signs (+). Any number of spaces or tab characters can precede or follow a comma or a plus sign. For example:

  ```
  PRINT file-spec[,...]
  ```

  This shows that you can optionally enter a list of files as the parameter.

Note: Some commands regard the plus sign as a concatenator, not as a separator. The parameter descriptions in the *VMS DCL Dictionary* state how each command interprets commas and plus signs.

1.2.1 Specifying a File

File specifications are the most common type of parameter. DCL commands can accept input file specifications (files that are acted upon by a command) and output file specifications (files that are created by a command). For complete details on file specifications, see Chapter 3.

The command descriptions in the *VMS DCL Dictionary* describe how each command interprets file specifications. Command descriptions provide information about defaults for file specifications that are entered as parameters. They also indicate whether you can use wildcards.

1.3 Entering Qualifiers

The qualifier section of the command descriptions in the *VMS DCL Dictionary* shows the qualifiers that are accepted by each command. It also indicates whether or not a qualifier accepts a value. Although typing any qualifier is optional, certain defaults are automatically applied. You need to be aware of the defaults that apply for each qualifier. The following sections describe types of qualifiers and qualifier defaults.

1.3.1 Types of Qualifiers

The three types of qualifiers are as follows:

- Command qualifiers modify commands. You can place a command qualifier anywhere in the command line. For example:

  ```
  $ PRINT/HOLD FARM.DAT
  $ PRINT FARM.DAT/HOLD
  ```

  The /HOLD qualifier is a command qualifier. Therefore, the two PRINT commands shown in this example are equivalent. The file FARM.DAT is placed in a hold state.

- Positional qualifiers can modify commands or parameters. They have different effects depending on where they are placed in the command line. If you place a positional qualifier after the command, it affects the entire command line. If you place a positional qualifier after a parameter, it affects only that parameter. For example:

  ```
  $ PRINT/COPIES=2 FARM.DAT,GROTON.DAT
  $ PRINT FARM.DAT/COPIES=2,GROTON.DAT
  ```

  The first PRINT command prints two copies of each of the files FARM.DAT and GROTON.DAT. The second PRINT command prints two copies of FARM.DAT, but only one copy of GROTON.DAT. You can use positional qualifiers after more than one parameter in the command line.

- Parameter qualifiers can be used only after a certain type of parameter. For example, the BACKUP command accepts several parameter qualifiers that apply only to input and output file specifications.

1.3.2 Qualifier Defaults

A qualifier default is defined as what happens when you omit the qualifier from the command line. Qualifiers can be either present or absent by default. The following paragraphs explain the syntax used in the *VMS DCL Dictionary* to display qualifiers and defaults.

Qualifiers can take one of the following formats:

- Qualifiers with positive and negative forms. These qualifiers have a value of true or false. You indicate a true value by naming the qualifier. You indicate a false value by inserting the prefix NO.

For example, the /LOG qualifier can be expressed positively or negatively. If you omit the qualifier, the default action is /NOLOG. The following shows how the /LOG qualifier is listed in a command description.

```
/LOG
/NOLOG (default)
```

- Qualifiers that require values. If you use a qualifier that requires a value, you must specify a value. If you omit the qualifier, then the default value is applied.

  For example, if you use the /COPIES qualifier, you must provide a numeric value. If you omit the /COPIES qualifier, the default is /COPIES=1. The following shows how the /COPIES qualifier is listed in a command description.

  ```
  /COPIES=n
  ```

  Qualifiers can also accept other types of values. For more information, see Section 1.3.3.

- Qualifiers that affect command execution only if explicitly present. There is no corresponding default.

  For example, the /ABORT qualifier does not affect the command if it is not specified. The following shows how the /ABORT qualifier is listed in a command description.

  ```
  /ABORT
  ```

- Qualifiers that override other qualifiers. Sometimes a command has a qualifier that is automatically applied as a default. Other qualifiers are available to override the default qualifier.

  For example, there are four access modes within the VMS operating system. They are user mode (least privileged), supervisor mode, executive mode, and kernel mode (most privileged). By default, all DCL commands run in supervisor mode. The DEFINE command lets you determine which mode it will run in. It has a /SUPERVISOR_MODE qualifier that applies supervisor mode as the default. However, you can override this default with either the /USER_MODE or EXECUTIVE_MODE qualifiers.

  The following example shows how these qualifiers are listed in a command description:

  ```
  /EXECUTIVE_MODE
  /SUPERVISOR_MODE (default)
  /USER_MODE
  ```

The command line is processed from left to right. If two or more contradictory qualifiers are present, then the right-most qualifier overrides the others. For example:

```
$ PRINT MYFILE/COPIES=3/BURST/COPIES=2/NOBURST
```

For this PRINT command, only the /COPIES=2 and the /NOBURST qualifiers are accepted.

Some commands contain conflicting qualifiers that cannot be specified in the same command line. If you use incompatible qualifiers, the command interpreter usually displays an error message. The command descriptions in the *VMS DCL Dictionary* indicate which qualifiers cannot be used together.

1.3.3 Qualifiers That Accept Values

Qualifiers can accept the following types of values:

- Keywords
- File specifications
- Character strings
- Numeric values

When you enter a value for a qualifier, separate the qualifier and the value with either an equal sign (=) or a colon (:). For example, the following expressions are equivalent:

/COPIES=3	/COPIES:3
/OUTPUT=DBA1:NEW.DAT	/OUTPUT:DBA1:NEW.DAT
/OVERRIDE=EXPIRATION	/OVERRIDE:EXPIRATION

Some qualifier keyword values require additional data. In these cases, separate the keyword from its data with a colon or an equal sign. For example, the following expressions are equivalent:

```
/PROTECTION:GROUP:RW
/PROTECTION=GROUP=RW
/PROTECTION=GROUP:RW
/PROTECTION:GROUP=RW
```

The command descriptions in the *VMS DCL Dictionary* use the following notation to indicate that a qualifier can accept a value or a list of values:

/qualifier=(value[, . . .])	The qualifier requires a value or list of values. If you specify a single value, you can omit the parentheses. However, if you specify a list of values, you must separate the values with commas and enclose them in parentheses. For example:
	
	`$ SET COMMAND/DELETE=TYPE`
	`$ SET COMMAND/DELETE=(PRINT,COPY)`

/qualifier[=(value[, . . .])]	The square brackets indicate that the value or list of values is optional. If you specify a single value, you can omit the parentheses. However, if you specify a list of values, you must separate the values with commas and enclose them in parentheses. For example:
	
	`$ RUNOFF/MESSAGES`
	`$ RUNOFF/MESSAGES=(OUTPUT,USER)`

To specify multiple keywords that require values, enclose the list in parentheses and separate the keyword and value with either an equal sign (=) or a colon (:). For example:

```
/BLOCKS=(START:0,END:10)
/PROTECTION=(OWNER=RWD,GROUP=RW)
```

If a qualifier accepts a keyword value, you can abbreviate the keyword as described in Section 1.3.5.

1.3.4 Qualifiers That Create Output Files

With some qualifiers you can specify the output file. For example, the /LIST and /OBJECT qualifiers let you specify the names of the files that result from running a language compiler. You can also use the /EXECUTABLE qualifier for the linker for the same purpose.

The default output file specification created by these qualifiers depends on the placement of the qualifier in the command. The following rules apply:

- If the qualifier is present by default, the output file specification defaults to the current default device and directory and the name of the first input file. The qualifier provides a default file type. Following are some examples:

Command	Output File
LINK A,B	A.EXE
LINK [TEST]A,[ ]B	A.EXE

In these examples, the qualifier /EXECUTABLE is present by default. The first input file name is A. The /EXECUTABLE qualifier provides a default file type of EXE. Therefore, the output file is named A.EXE and is written to the current default device and directory.

- If the qualifier is used after the command name and does not include an output file specification, the output file specification defaults to the current default device and directory and the name of the first input file. The qualifier provides a default file type. Following is an example:

Command	Output File
LINK/EXECUTABLE A,B	A.EXE

- If the qualifier is used after an input file specification and does not include an output file specification, the output file specification defaults to the device, directory, and file name of the immediately preceding input file. The qualifier provides a default file type. Following are some examples:

Command	Output Files
MACRO A+C/LIST	A.OBJ and C.LIS
LINK A+[TEST]D/EXECUTABLE	[TEST]D.EXE

In the first example, the plus sign (+) concatenates the files A.MAR and C.MAR (MAR is the file type for a MACRO file). The /OBJECT qualifier is present by default, so MACRO creates an object file with the same name as the first input file. The /LIST qualifier is placed after C.MAR. It affects only that file and causes the listing to be named C.LIS.

In the second example, the LINK command links two object files together. Because the /EXECUTABLE qualifier is specified after the second input file, the resulting output file is named [TEST]D.EXE. This overrides the default output file naming convention.

- If the qualifier specifies a file specification for the output file, then any fields entered in the file specification are used to name the output file. Missing device and directory fields are filled in by the current defaults. Following are some examples:

Command	Output Files
LINK A+B/EXECUTABLE=C	C.EXE
MACRO/LIST=FRED B+C	FRED.LIS and B.OBJ. Both files contain B and C concatenated.

In the first example, A and B are linked together. The /EXECUTABLE qualifier is used to override the default and name the output file C.EXE.

In the second example, the /LIST qualifier is used to name the listing file FRED.LIS. The /OBJECT qualifier is present by default so the object file B.OBJ is also produced.

- If the current default device or directory is different from that specified in the command parameter, the current default is used for the output file.

  If the current default device or directory is different from that specified for a command parameter, and if the output qualifier does not include a specification, the output file resulting from the qualifier takes its device and directory from the command parameter. In the following examples the current default device and directory is DISK1:[JONES]:

Command	Output Files
MACRO DISK2:[SMITH]A /LIST=CAT	DISK1:[JONES]A.OBJ and DISK1:[JONES]CAT.LIS
MACRO [.PROGRAMS]A/LIST	DISK1:[JONES]A.OBJ and DISK1:[JONES.PROGRAMS]A.LIS

In the first example A.MAR is located in DISK2:[SMITH]. Because a file specification is included with the /LIST qualifier, the LIS file is placed in the current default device and directory, DISK1:[JONES].

In the second example, A.MAR is located in DISK1:[JONES.PROGRAMS]. Because a file specification is not included with the /LIST qualifier, the LIS file is placed in the directory specified by the command parameter.

In all cases, the version number of the output file is always one greater than any existing file with the same file name and file type.

1.3.5 Abbreviating Qualifiers and Keywords

When the DCL command interpreter examines a command line, it looks at only the first four characters of a qualifier name. This means that you can abbreviate any qualifier by typing only the first four characters. All qualifier names are unique within four characters. You can abbreviate a qualifier to less than four characters, as long as the abbreviation remains unique. Follow these rules:

- Disregard the slash character (/) in counting characters.
- Count the underscore. For example, /BY_OWNER can be shortened to /BY_O.
- Many qualifiers permit a negative form. In applying the minimum four-character truncation rule, count the NO prefix as the first one of the four characters. For example, /NOLIST is the negative form of the /LIST qualifier. In this case, the minimum truncation that guarantees uniqueness is /NOLIS.

The command interpreter looks at keywords in their entirety. You can abbreviate keywords to the fewest number of characters that provide a unique abbreviation within a set of possible keywords.

1.3.6 Commonly Used Qualifiers

Table 1–2 lists qualifiers that are used with many DCL commands.

Table 1–2 Commonly Used Qualifiers

Qualifier	What It Does
/BACKUP	Indicates that the command should operate on files or directories backed up before or after the specified date. For example: `$ DELETE/BACKUP/BEFORE=15-APR *.*;*`
/BEFORE[=time]	Indicates that the command should operate on files or directories dated before the specified time. For example: `$ DELETE/BEFORE=YESTERDAY *.*;*`
/BY_OWNER[=uic]	Indicates that the command should operate on files or directories belonging to the specified owner. For example: `$ DELETE/BY_OWNER=[BAND,ROHBA] *.*;*`
/CONFIRM	Requests your permission before performing the indicated operation. For example: `$ DELETE/CONFIRM *.*;*`

Table 1–2 (Cont.) Commonly Used Qualifiers

Qualifier	What It Does
/CREATED	Indicates that the command should operate on files or directories created before or after the specified date. For example: `$ DELETE/CREATED/SINCE=YESTERDAY *.*;*`
/EXCLUDE	Indicates that the command should not operate on the specified files. For example: `$ PRINT/EXCLUDE=TAX.DAT *.DAT`
/FULL	Requests detailed information. For example: `$ DIR/FULL`
/LOG	Displays each file specification affected by the command. For example: `$ DELETE/LOG *.*;*`
/MODIFIED	Indicates that the command should operate on files or directories modified before or after the specified date. For example: `$ DELETE/MODIFIED/SINCE=YESTERDAY *.*;*`
/NOOUTPUT	Suppresses command output. However any value that results from the operation is still returned. For example: `$ DIR/NOOUTPUT`
/OUTPUT[=file-spec]	Redirects the command output to a file. Default file names and types vary with different commands. For example: `$ SHOW QUEUE/OUTPUT=MYJOBS.DAT`
/PROTECTION=(code)	Specifies the UIC-based protection to be applied. For example: `$ CREATE/DIRECTORY/PROTECTION=(S:RWED,O:RWED,G:R)`
/SINCE[=time]	Indicates that the command should operate on files or directories dated after the specified time. For example: `$ DELETE/SINCE=YESTERDAY *.*;*`

1.4 Entering Dates and Times

Certain commands and qualifiers accept date and time values. You can specify these values in one of the three following formats:

- Absolute time
- Delta time
- Combination time

The command descriptions in the *VMS DCL Dictionary* indicate the time formats accepted by commands and qualifiers.

1.4.1 Absolute Time

Absolute time is a specific date or time of day. The format for an absolute time is as follows:

[dd-mmm-yyyy[:]][hh:mm:ss.cc]

The fields are as follows:

Field	Meaning
dd	Day of the month; an integer in the range of 1-31
mmm	Month; specified as JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, or DEC
yyyy	Year; an integer
hh	Hour of the day; an integer in the range of 0-23
mm	Minute of the hour; an integer in the range of 0-59
ss	Seconds; an integer in the range of 0-59
cc	Hundredths of a second; an integer in the range of 0-99

You can specify an absolute time value using either the date or the time, or both. Follow these rules:

- If you specify both the date and the time, type a colon between them.
- If you specify the date, include the hyphen (-).
- You can truncate either the date or the time on the right.
- You can omit any of the fields within the date or time, as long as you type the punctuation marks that separate the fields.
- When fields are omitted, the system supplies default values. If you omit a date field, the default is the corresponding field for the current date. If you omit a time field, the default value is zero.
- If you specify a past time in a command that expects the current or a future time, the current time is used.
- You can use one of the following keywords to specify an absolute time.

Keyword	Meaning
TODAY	The current day, month, and year at 00:00:00.0 o'clock
TOMORROW	24 hours after 00:00:00.0 o'clock today
YESTERDAY	24 hours before 00:00:00.0 o'clock today

Some examples of valid absolute time specifications follow:

Time Specification	Result
31-DEC-1988:12	12:00 noon on December 31, 1988
15	3:00 P.M., today
15-::30	12:30 AM, on the 15th day of the current month
31-DEC	Midnight (00:00 o'clock) at the beginning of the 31st of December, this year
15-	The 15th day of the current month and year, at midnight
18:30	6:30 P.M., today
00:00:00.2	Two-hundredths of a second after midnight, today

1.4.2 Delta Time

Delta time is an offset from the current date and time to a time in the future. The format for a delta time is as follows:

[dddd-][hh:mm:ss.cc]

The fields are as follows:

Field	Meaning
dddd	Number of days; an integer in the range of 0-9999
hh	Number of hours; an integer in the range of 0-23
mm	Number of minutes; an integer in the range of 0-59
ss	Number of seconds; an integer in the range of 0-59
cc	Number of hundredths of seconds; an integer in the range of 0-99

Follow these rules for entering a delta time:

- If you specify both the number of days and the time, type a hyphen between them.
- If you specify the number of days, include the hyphen (-).
- You can truncate the time on the right.
- You can omit any of the fields within the time, as long as you type the punctuation marks that separate the fields.
- When a field is omitted, the system supplies default values. If you omit a time field, the default is zero.

Some examples of valid delta time specifications follow:

Time Specification	Result
3-	3 days from now (72 hours)
3	3 hours from now
:30	30 minutes from now
3-:30	3 days and 30 minutes from now
15:30	15 hours and 30 minutes from now

1.4.3 Combination Time

To combine absolute and delta time, specify an absolute time plus (+) or minus (-) a delta time. The format is as follows:

"[absolute time][+delta time]"
or
[absolute time][-delta time]

The fields are as follows:

Field	Meaning
absolute time	Use the syntax for entering an absolute time value: [dd-mmm-yyyy[:]][hh:mm:ss:cc]
delta time	Use the syntax for entering a delta time value: [dddd-][hh:mm:ss:cc]

The rules for entering an absolute or a delta time value still apply. In addition, do the following:

- Precede the delta time value by a plus or minus sign. (Note that the minus sign is the same keyboard key as the hyphen.)
- Whenever a plus sign precedes the delta time value, enclose the entire time specification in quotation marks.
- You can omit the absolute time value. If you do, the delta time is offset from the current date and time.
- Specify date and time information as completely as possible.

Some examples of valid combination time specifications follow:

Time Specification	Result
"+5"	Current time plus 5 hours. The absolute time portion is omitted, so it defaults to the current date and time.
−1	Current time minus 1 hour. The minus sign (−) indicates a negative offset. (The 1 is interpreted as an hour, not a day, because it is not followed by a dash.)
"+:5"	Current time plus 5 minutes. The absolute time portion is omitted, so it defaults to the current date and time.
−:5	Current time minus 5 minutes. The absolute time specification is omitted, so it defaults to the current date and time.
−1-00	Current time minus 1 day. The minus sign (−) indicates a negative offset. The dash (-) separates the day from the time field.
"31-DEC:+:5"	12:05 AM on December 31 of the current year. The absolute time specification (before the colon) defaults to midnight on December 31 of the current year. The plus sign (+) indicates a positive offset.
31-DEC:−00:10	11:50 PM on December 30 of the current year. The absolute time specification (before the colon) defaults to midnight on December 31 of the current year. The minus sign (−) after DEC: indicates a negative offset. The dash (-) indicates that the value for the day is missing.

Note: If a description of a qualifier states that the value can be expressed as an absolute time, a delta time, or a combination of the two, you must specify a delta time as if it were a part of a combination time. For example, to specify a delta time value of five minutes from the current time, use "+:5" (not ":5").

2 Editing the DCL Command Line

At the DCL command level, you can use many individual keys and key combinations to change what you type, recall commands, or display information. DCL also provides you with the option of defining keys so that you can enter commands with fewer keystrokes. This chapter describes how to use the keyboard in DCL.

2.1 Executing a DCL Command

The RETURN key is recognized as a line terminator. Once you type a command at the DCL prompt, press RETURN to terminate the line and send it to the DCL command interpreter for execution. CTRL/Z and certain escape sequences are also recognized as line terminators.

2.2 Interrupting and Canceling a DCL Command

After you enter a DCL command, you can temporarily interrupt its execution, run other commands, and then return to executing the command that was interrupted. To interrupt the execution of a command, use CTRL/T, CTRL/Y, or CTRL/C. These keys behave in different ways depending upon the type of DCL command that is currently executing.

As mentioned in Chapter 1, there are built-in commands and command images. A built-in command is a part of the command interpreter. A command image is a program that is called by the command interpreter.

A command image can be privileged or nonprivileged. Privileged command images are those installed as such by VMS, you, or the system manager. They may vary from system to system. Whether a command image is privileged or nonprivileged affects the way it reacts to interruptions. Typically, you will not know if a command is privileged or not until you interrupt it and try to execute certain commands. For more information on privileged command images, see the *VMS Install Utility Manual.*

The following tables list the results of CTRL/T, CTRL/Y, and CTRL/C on the different types of DCL commands.

Table 2–1 Interrupting Built-in Commands

Key Combination	Result
CTRL/T	Interrupts the command, displays a line of information (node name, process name, current time, image name, elapsed CPU time, page faults, direct and buffered I/O operations, and pages in physical memory) and then resumes command execution. Use CTRL/T at the DCL prompt whenever you want to display any of this information. SET CONTROL=T must be set, either by you or by the system manager. CTRL/T will not echo information if the system is temporarily hung or if your terminal is set to NOBROADCAST.
CTRL/Y	No effect.
CTRL/C	Cancels command execution.

Table 2–2 Interrupting Nonprivileged Command Images

Key Combination	Result
CTRL/T	Same as for built-in commands.
CTRL/Y	Interrupts the command. The interrupted command is temporarily suspended and control returns to the command interpreter. You see the DCL prompt. After interrupting a nonprivileged command image with CTRL/Y, you can do the following: • Type CONTINUE to resume execution of the interrupted command. • Enter another command. Entering any number of built-in commands does not affect the interrupted one. (All the DCL built-in commands are listed in Chapter 1.) Entering any other type of command effectively cancels the interrupted one. • Type STOP to terminate immediately the interrupted command. STOP suppresses any cleanup activities such as the display of error messages.
CTRL/C	CTRL/C works like CTRL/Y unless the program you are executing has enabled CTRL/C as the cancel key (which is a common practice).

Table 2–3 Interrupting Privileged Command Images

Key Combination	Result
CTRL/T	Same as for built-in commands.
CTRL/Y	Interrupts the command. The interrupted command is temporarily suspended and control returns to the command interpreter. You see the DCL prompt. After interrupting a privileged command image with CTRL/Y, you can do the following: • Type CONTINUE to resume execution of the interrupted command. • Enter another command. Entering SPAWN, ATTACH, or CONTINUE followed by any other command does not affect the interrupted one. Entering any other command effectively cancels the interrupted one. • Type STOP to terminate immediately the interrupted command. STOP suppresses any cleanup activities such as the display of error messages.
CTRL/C	CTRL/C works like CTRL/Y unless the program you are executing has enabled CTRL/C as the cancel key (which is a common practice).

After interrupting a command image you can use the SPAWN command (a built-in DCL command). The SPAWN command creates a subprocess. Within that subprocess you can execute another command image without losing the context of the interrupted one. Log out of a subprocess and type CONTINUE to return to what you were doing.

For example, you can interrupt the EDT editor with the SPAWN command to create a subprocess, read your mail, return to your main process, and continue your editing session, as follows:

```
$ EDT WORKFILE.TXT
   .
   .
   .
CTRL/Y
$ SPAWN
%DCL-S-SPAWNED, process MOOLA_1 spawned
%DCL-S-ATTACHED, terminal now attached to process MOOLA_1
$ MAIL
MAIL>
   .
   .
   .
MAIL> EXIT
$ LOGOUT
  Process MOOLA_1 logged out at 31-DEC-1988 17:03:04.03
%DCL-S-RETURNED, control returned to process MOOLA
$ CONTINUE
```

After you type the CONTINUE command, you can press CTRL/W to refresh the screen.

2.3 Editing Commands

There are many types of terminals, each with its own operating characteristics. In general, they all have standard function and line editing keys. This section describes the line editing keys that let you edit the DCL command line.

To use the line editing keys, line editing must be enabled on your terminal. To see whether or not line editing is enabled, type the SHOW TERMINAL command. The current characteristics of the terminal are displayed. The current status of line editing is displayed in the first column under "Terminal Characteristics." For example:

```
$SHOW TERMINAL

Terminal: _VTA130:    Device_Type: VT200_Series  Owner: ROHBA
LAT Server/Port: L121/Port_3
Physical terminal: _LTA130:
   Input:   9600      LFfill:  0      Width:  80      Parity: None
   Output:  9600      CRfill:  0      Page:   24
Terminal Characteristics:
   Interactive         Echo                Type_ahead          No Escape
   No Hostsync         TTsync              Lowercase           Tab
   Wrap                Scope               No Remote           No Eightbit
   Broadcast           No Readsync         No Form             Fulldup
   No Modem            No Local_echo       No Autobaud         Hangup
   No Brdcstmbx        No DMA              No Altypeahd        Set_speed
   No Line Editing     Insert editing      No Fallback         No Dialup
   No Secure server    Disconnect          No Pasthru          No Syspassword
   No SIXEL Graphics   No Soft Characters  No Printer Port     Numeric Keypad
   ANSI_CRT            No Regis            No Block_mode       Advanced_video
   No Edit_mode        DEC_CRT             No DEC_CRT2
```

If you find that line editing is not enabled, type the SET TERMINAL/LINE_EDIT command to enable it.

You can edit a command line in either overstrike or insert mode. In overstrike mode, the character you type overwrites the character indicated by the cursor. In insert mode, the character you type is inserted to the left of the cursor. The SET TERMINAL command establishes the default editing mode. You can use the SET TERMINAL/INSERT command or the SET TERMINAL/OVERSTRIKE command. Press CTRL/A to change editing modes (CTRL/A acts as a toggle).

You can use the SET TERMINAL/WRAP command so that whenever you enter more characters than will fit on one line of the terminal screen, the text wraps to the next line. Keep in mind that you can edit only the line where your cursor appears. When text wraps, you cannot use the up arrow key to move the cursor up to edit the previous line. However, you can move the cursor up to the previous line by using the DELETE key to delete all the characters in the current line.

By default, changes made with the SET TERMINAL command apply only to the current session. You can include SET TERMINAL commands in your LOGIN.COM file to set the terminal to operate the way you want. For more information on the SET TERMINAL command, see the *VMS DCL Dictionary*. For more information on LOGIN.COM, see the *Introduction to VMS*.

2.3.1 Deleting Characters

The DELETE key backspaces over the most recently entered character and deletes it. On a hardcopy terminal, the deleted letters are displayed between backslash characters so you can see what is being deleted. On a video display terminal, pressing the DELETE key erases the character from the screen and moves the cursor backwards. Note that the key that performs the delete function is marked RUBOUT on some terminals and is marked <X] on LK201 keyboards.

Note: Do not use the BACKSPACE key to delete characters when you are entering DCL commands; the BACKSPACE key does not delete characters.

2.3.2 Deleting Lines

When line editing is enabled, you can use CTRL/U to delete characters from the beginning of the line to the current cursor position.

When line editing is not enabled, you can use CTRL/U to cancel an entire line. When you cancel a line with CTRL/U, the system ignores the line and redisplays the DCL prompt.

2.3.3 More Editing Commands

Use the keys listed in Table 2–4 to edit the DCL command line. For some of these keys to work, line editing must be enabled.

Table 2–4 Line Editing Keys

Key	Function
DELETE	Deletes the last character entered at the terminal. (On some terminals, the DELETE key is labeled RUBOUT.) The DELETE key also works when line editing is disabled.
CTRL/A and F14[1]	Switches between overstrike mode and insert mode. The default mode (as set with the SET TERMINAL /LINE_EDIT command) is reset at the beginning of each line.
CTRL/D and Left arrow	Moves the cursor one character to the left.
CTRL/E	Moves the cursor to the end of the line.
CTRL/F and Right arrow	Moves the cursor one character to the right.
CTRL/H and BACKSPACE and F12[1]	Moves the cursor to the beginning of the line.

[1]This key is available only on an LK201 keyboard.

Table 2–4 (Cont.) Line Editing Keys

Key	Function
CTRL/I and TAB	Moves the cursor to the next tab stop on the terminal. The system provides tab stops at every eighth character position on a line. Tab settings are hardware terminal characteristics that, in general, you can modify. The TAB key also works when line editing is disabled.
CTRL/J and LINEFEED and F13[1]	Deletes the word to the left of the cursor.
CTRL/U	Deletes characters from the beginning of the line to the cursor. (This overrides the standard CTRL/U function, which deletes the current line.)
CTRL/V	Turns off some of the line editing function keys. For example, if you press CTRL/V followed by CTRL/D, a CTRL/D is generated instead of the cursor moving left one character. CTRL/D is a line terminator at the DCL level. When combined with CTRL/V, characters that are not line terminators have no effect. Examples are CTRL/H and CTRL/J. However, certain control keys, such as CTRL/U, retain their line editing function in spite of CTRL/V.
F7,F8,F9,F11	Reserved to DIGITAL

[1]This key is available only on an LK201 keyboard.

2.4 Recalling Commands

At the DCL command level you can recall previously typed command lines by using one of the following:

- CTRL/B
- Up and down arrow keys
- RECALL command

All of the above let you display the commands that are stored in the recall buffer. The recall buffer holds up to 20 previously entered commands. Once a command is displayed, you can reexecute it or edit it.

Pressing CTRL/B displays the previous command line. Pressing CTRL/B again displays the line before the previous line, and so on to the last saved command line.

Pressing the up and down arrow keys recalls the previous and successive command respectively. You can press the arrow keys repeatedly to move through the commands.

The RECALL/ALL command displays a list of previously entered commands. You can then request a command by its number. For example:

```
$ RECALL 3
```

After you press RETURN, the third command from the list is displayed at the DCL prompt. (The RECALL command itself is not placed in the buffer.)

You can also follow RECALL with the first character(s) of the command line you want to display. RECALL scans the previous command lines (beginning with the most recent one) and returns the first one that begins with the character(s) you typed. For example:

```
$ RECALL D
$ DIR
```

At the DCL command level, you can always use CTRL/B and the up and down arrow keys for command recall. If you are in a utility or an application program that uses VMS screen management software, you can also use these keys to perform command recall. Line editing must be enabled. In this case, you can recall up to the last 20 command lines. Examples of utilities that have this feature are MAIL, DEBUG, SHOW CLUSTER, the System Dump Analyzer (SDA), and the VAXTPU editor.

2.5 Terminal Function Keys

The terminal has standard function keys that let you perform terminal functions. Table 2–5 lists the standard terminal function keys and describes their use.

Table 2–5 Terminal Function Keys

Key	Function
CTRL/B and Up arrow	Displays the last command line entered. If pressed again, displays the previous command in the recall buffer. The recall buffer stores the 20 most recently entered commands.
CTRL/C and F6[1]	During command entry, cancels command processing. CTRL/C is displayed as "Cancel".
	Certain applications enable CTRL/C as the cancel key. For these applications, CTRL/C cancels the operation in progress. If CTRL/C is not enabled, the action is changed to an interrupt (CTRL/Y).
CTRL/K	Advances the current line to the next vertical tab stop.
CTRL/L	Causes the terminal to go to the beginning of the next page. This use of this key is ignored when line editing is enabled.
CTRL/O	Alternately suspends and continues display of output to the terminal. CTRL/O is displayed as "Output off" and "Output on".
CTRL/Q	Restarts terminal output that was suspended by CTRL/S.

[1]This key is available only on an LK201 keyboard.

Table 2–5 (Cont.) Terminal Function Keys

Key	Function
CTRL/R	Retypes the current input line and leaves the cursor positioned at the end of the line.
CTRL/S	Suspends terminal output until CTRL/Q is pressed.
CTRL/T	Momentarily interrupts terminal output to display a line of statistical information about the current process. SET CONTROL=T must be specified in the system-wide login command file, or by you (in LOGIN.COM or interactively). The display includes your node and user name, the time, the name of the image you are running, and information about system resources used during your current terminal session. You can also use the CTRL/T key to determine if the system is operating. (CTRL/T does not echo information if the system is temporarily hung or if your terminal is set to NOBROADCAST.)
CTRL/U	Cancels the current input line.
CTRL/X	Cancels the current line and deletes data in the type-ahead buffer.
CTRL/Y	Interrupts command processing. CTRL/Y is displayed as "Interrupt". You can disable CTRL/Y with the command SET NOCONTROL=Y. Under most conditions, CTRL/Y returns you to the DCL prompt. The program running is still active. You can enter any of the commands discussed in Section 2.2 and then continue the program with the CONTINUE command.
CTRL/Z and F10[1]	Signals the end of the file for data entered from the terminal. CTRL/Z is displayed as "Exit".
Down arrow	Displays the next line in the recall buffer.
RETURN	Sends the current line to the system for processing. (On some terminals, the RETURN key is labeled CR.) Before a terminal session, initiates a login sequence.

[1]This key is available only on an LK201 keyboard.

2.6 Defining Terminal Keys

Key definitions let you customize your keyboard so you can enter DCL commands with fewer keystrokes. A key definition is a string of characters that you assign to a particular terminal key. When a key is defined, you can press it instead of typing the characters. A key definition usually contains all or part of a command line. When you press a defined key, the command is either executed or displayed on your terminal.

Some definable keys are enabled for definition all the time (like keys PF1 through PF4 and keys F17 through F20 on VT200 series terminals). Other keys, including KP0 through KP9, PERIOD, COMMA, and MINUS, need to be enabled for definition purposes. Enter either the SET TERMINAL /APPLICATION command or the SET TERMINAL/NONUMERIC command before using these keys. For a list of keys that you can define and for more information on how to create definitions, see the description of the DEFINE /KEY command in the *VMS DCL Dictionary*.

3 File Specifications

This chapter provides the rules for using VMS file specifications and device names with DCL commands. It contains information about the following:

- Specifying the parts of a file specification; including the node, device, directory, file name, and file type
- Using wildcards in file specifications to indicate groups of files

3.1 Format for File Specifications

To uniquely identify a file that you want to access, use a file specification. A file specification provides the system with all the information it needs to identify a file. A full file specification has the following format:

node::device:[directory]filename.type;version

The fields are as follows:

node	A network node name; applicable only to systems that support DECnet–VAX.
device	The name of the physical device on which the file is stored or is to be written.
directory	The name of the directory under which the file is cataloged. Square brackets ([ ]) or angle brackets (< >) can be used to delimit directory names.
filename	The name of the file. It can have up to 39 alphanumeric characters, including the hyphen and the underscore.
type	Identification of the structure or the type of data in the file. It can have up to 39 alphanumeric characters, including the hyphen and the underscore.
version	The version number of the file. Versions are identified by a decimal number, which is incremented by 1 each time a new version of the file is created.

Some of the information in a file specification can be provided using defaults, as discussed in Section 3.7.

The following is an example of a file specification:

```
BOSTON::DISK1:[LICENSES]SUMMER_1984.DAT;18
|      |      |         |         |   |
node   device directory filename   type version
```

Observe the following rules when entering a file specification:

- The maximum size of a file specification, including all delimiters, is 255 characters.
- Punctuation marks and brackets are required to separate the fields of the file specification.

- The directory field applies only to files on disks (as opposed to files on tape).
- The node field is used only if your system is part of a network.
- The fields for file name, type, and version apply only to files on mass storage devices (such as disks and tapes).
- Only the device field is used in the file specification for record-oriented devices (such as printers and card readers).

Each field is described in greater detail in the following sections.

3.2 Network Nodes

A node is an individual system that is part of a computer network. If your system is part of a network, the node that you access when you log in is your local node. Other nodes in the network are remote nodes. Use a node name when you want to specify a file on a remote node.

A node specification has the following format:

```
node["access-control-string"]::
```

Observe the following rules when entering a node name as part of a file specification:

- A node name can contain 1 to 6 alphanumeric characters and must contain at least one alphabetic character.
- It must always be followed by a double colon (::).
- When you specify a node name, you can optionally include a 0 to 42-character *access control string*. An access control string contains login information to be sent to the remote node. For more information on access control strings, see Section 3.2.2.
- You can use a logical node name in place of the node name. For information on logical node names, see Chapter 4.

When you access a file on a remote node, DECnet–VAX logs in at the remote node. To do this, the system needs login information for that node. You can supply it with an access control string. If you omit the access control string, the login information sent to the remote node is determined as follows:

- If a proxy login account exists for you on the remote node, then the system logs you in using that account. (A proxy login account allows selected users to log in to a node.)
- If a proxy login account does not exist, the system uses the default DECnet account for that node as specified by the local system manager.

If you include an access control string, the system uses it to log you into the remote node. The remainder of the file specification is passed to the remote node and is interpreted there.

If you specify a local node as part of a file specification, the system logs you in over the network to perform the file operation, even though the file exists on your local node.

3.2.1 Network File Specifications

There are three formats for network file specifications. In each format, the node specification can optionally include an access control string.

- Conventional format for VMS files:

  node::device:[directory]filename.type;version

- Format used to provide a foreign file specification:

  node::"foreign-file-spec-string"

  A foreign file specification is a file that does not conform to the VMS syntax. For example:

  ```
  $ COPY BOSTON::"TEST?.DAT" *
  ```

  The file name above contains a question mark character (?), which is not recognized as a valid filename character in VMS. Therefore, the file name must be enclosed in quotes. It must also be in a format that is recognized by the operating system of the remote node you are accessing.

Note: There are some restrictions when you copy files to or from an ULTRIX system. For more information, see the *VMS Convert and Convert /Reclaim Utility Manual.*

- Format used to indicate a task specification string:

  node::"task-spec-string"

  A task specification string identifies a program to be executed on the remote node. For example:

  ```
  BOSTON::"TASK=TEST2"
  ```

  The above specification identifies the program TEST2 on the remote node BOSTON. Usually, task specification strings are used within a program to enable it to communicate with another program on a remote node.

For more information, see the *VMS Networking Manual.*

3.2.2 Access Control Strings

The access control string designates an account you can log into on the remote node. A node name with an access control string has the following format:

node"access-control-string"::

Enclose the access control string in quotation marks and follow it with a double colon (::).

For VMS systems, the access control string consists of a user name, followed by one or more spaces or tabs and a password. For example:

```
$ DIR BOSTON"HIGGINS DUKE"::WEASEL2:[BORIS]ACCOUNTS.DAT
```

Here, BOSTON is the network node name. "HIGGINS DUKE" is an access control string where:

- HIGGINS is a user name on the node BOSTON.

- DUKE is the password associated with that name.

3.3 Devices

There are two classifications for devices in VMS:

- Mass storage devices—they save the contents of files on a magnetic medium. Files that are saved can be accessed at any time and updated, modified, or reused. Disks and magnetic tapes are mass storage devices.
- Record-oriented devices—they read and write only single physical units of data at a time and do not provide online storage of the data. Terminals, printers, mailboxes, and card readers are record-oriented devices. (Printers and card readers are also called *unit record devices.*)

When you specify a file that is not located on the current default device, you need to provide the name of the device where it is located. For files on disks, you must also specify the directory where the file is cataloged. In addition, some commands (such as ALLOCATE and MOUNT) require you to specify a device name. When a command requires a device name, include only the device portion of the file specification.

You can use physical, logical, or generic names to refer to devices. These types of device names are described in the following sections. In addition, if your system is part of a cluster, certain devices are accessible to all members of the cluster. Section 3.3.4 describes the syntax for cluster device names.

3.3.1 Device Names

Each physical device is uniquely identified by a device name. A device name has the following format:

ddcu

The fields are as follows:

dd	Device code. Table 3–1 lists some examples of valid device codes.
c	Controller designation. The controller designation, along with the unit number, identifies the location of the device within the hardware configuration of the system. Controller designations are alphabetic letters A through Z.
u	Unit number. The unit number, along with the controller designation, identifies the location of the device within the hardware configuration of the system. Unit numbers are decimal numbers 0 through 65,535.

Observe the following rules when entering a device name as part of a file specification:

- The maximum length of the device name, including the controller designation and the unit number, is 15 characters.

- A device name must be followed by a colon (:).

Table 3–1 Examples of Device Codes

Code	Device Type
DJ	RA60 disk
DU	RA80, RA81, RD52, RD53, RD54 disk
DX	RX01 floppy diskette
LP	Line printer on LP11
LT	Local area terminal
MB	Mailbox
MU	TA78, TA81, TK50, TU81 magnetic tape
NET	Network communications logical device
NL	System "null" device
OP	Operator's console
RT	Remote terminal
TT	Interactive terminal
TX	Interactive terminal
VT	Virtual terminal
XE	DEUNA synchronous communications line
XQ	DEQNA synchronous communications line

Whenever a disk or tape is mounted on a device, the system recognizes it as a *volume*. It also recognizes volume sets. A *volume set* consists of two or more related volumes. To access a file on a tape volume set, specify any device that has been allocated to it.

To access a file on a disk volume set, you have the following options:

- Specify the name of the device on which the first volume in the set is mounted.
- Specify the logical name that was assigned to the volume set when it was mounted.

In any case, if you do not specify a device name, the current default device name is supplied. For more information on volumes and volume sets, see the *Guide to VMS Files and Devices.*

Note: Some commands accept output file specifications. You can replace an output file specification with the name of a record-oriented device such as a printer or a terminal. For example:

```
$ COPY DISFILE.DAT TTB4:
```

The COPY command sends the file DISFILE.DAT to the terminal named TTB4. The terminal accepts and displays the file one record at a time. When you use a device name as a file specification, follow the device name with a colon (:).

3.3.2 Logical Device Names

The system manager can set up logical names for the available devices on the system. You should use these logical names when referring to devices. In doing this, you can achieve file and device independence. Using a logical device name lets you access a file, regardless of which physical device actually holds the disk or tape that contains the file. The system manager ensures that logical device names are always equated to the correct physical devices.

When you use a logical device name in a file specification, you must follow it with a colon. For example:

```
$ TYPE COD1:[NOAH]ANIMALS.LIS
```

COD1 is a logical device name for the device that contains the disk volume with the file [NOAH]ANIMALS.LIS. As long as the system manager defines the logical name COD1 correctly, the system can access the file, regardless of where the volume is mounted.

For more information on logical device names, see Chapter 4.

3.3.3 Generic Device Names

The MOUNT and ALLOCATE commands let you specify a *generic device name*. In a generic device name the controller designation (c) or the unit number (u) is not specified. When you use a generic device name, the system locates an available device unit whose physical name satisfies the portions of the generic device name that are specified. For example, if you enter the ALLOCATE command and specify only a device type, the ALLOCATE command locates an available device of that type.

If you specify a generic device name for any other command, the following defaults are applied:

- If you omit the controller designation, it is assumed to be A.
- If you omit the unit number, it is assumed to be 0.

3.3.4 Cluster Device Names

A cluster device name has the following format:

cluster-node$ddcu

where:

- *cluster-node* is the name of the node to which the device is attached.
- *ddcu* is the format for a device name as described in Section 3.3.1.

If a device is dual pathed (it is connected to two nodes), you should specify the cluster device name in the following format:

$allocation-class$ddcu

where:

- *allocation-class* is a value assigned to the nodes connecting a dual pathed device.

- *ddcu* is the format for a device name as described in Section 3.3.1.

For more information on cluster device names, see the *VMS VAXcluster Manual*.

3.4 Directories

A directory is a special type of file that catalogs (by name and location) a set of files on a disk volume or a disk volume set. A directory file contains the following information for every file cataloged within it:

- The file name, type, and version number
- A pointer to the file header, which describes the file (including owner, protection, attributes, and location on the disk)

Each directory has a file type of DIR. For example, PARTY.DIR;1 is a directory file. Because you cannot edit a directory file, all directory files have a version number of 1.

3.4.1 Directory Structure

Every disk contains a main directory that is set up by the system manager. This main directory is referred to as the *master file directory* (MFD). The MFD contains a list of all user file directories (UFDs) on the volume, with pointers to each UFD.

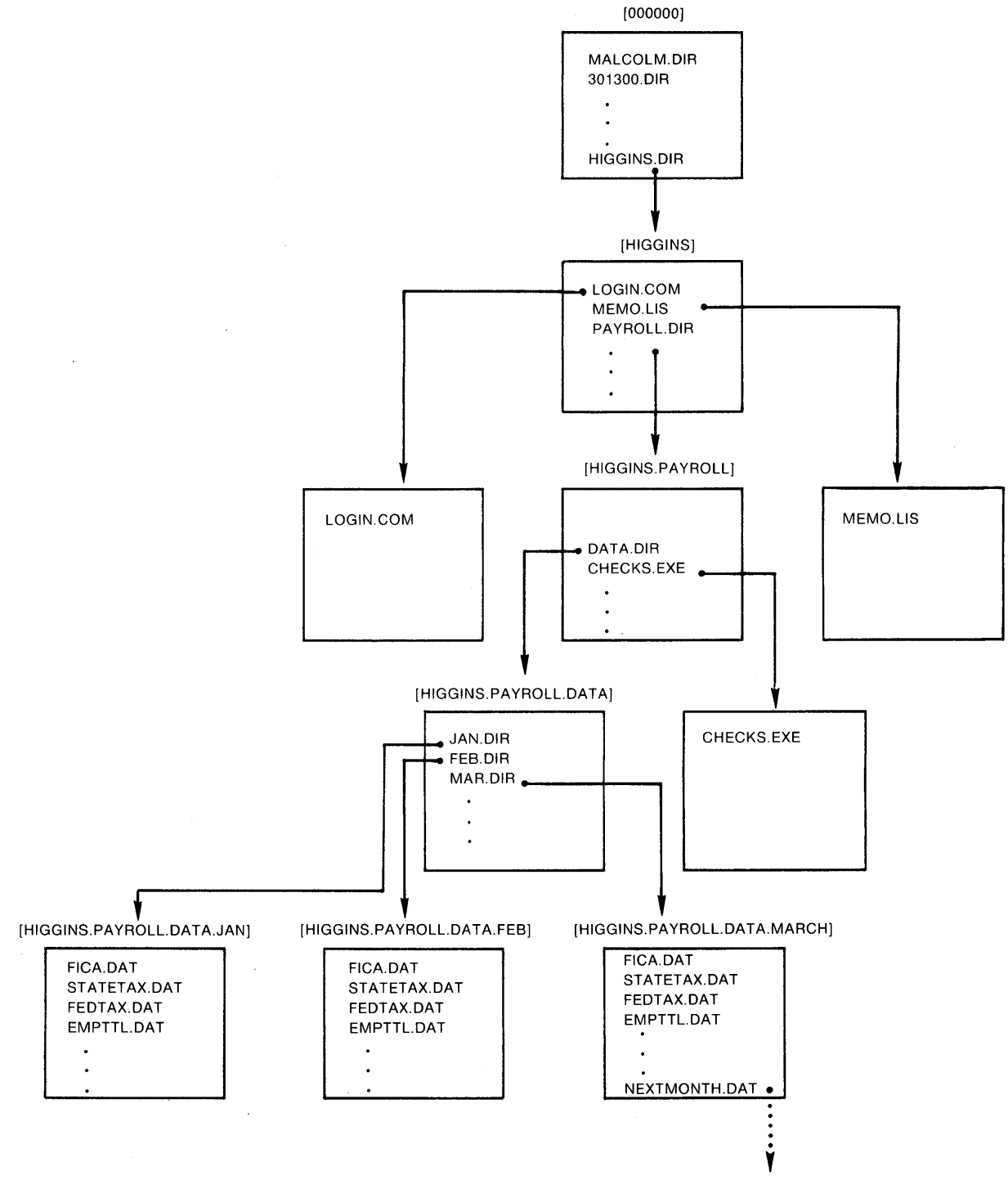
A *user file directory* (UFD) exists for each user on the system. It lists the names of files cataloged in the user's directory and has information pointing to each cataloged file. When you log into the system your default directory is your UFD, which is often referred to as your *top-level directory*. A UFD can hold data files and subdirectories.

The term *subdirectory* refers to any directory file that is not an MFD or a UFD. A subdirectory contains names and pointers for the files that are cataloged within it. It can contain an entry for another subdirectory; that subdirectory can contain an entry for another subdirectory, and so on up to seven levels of subdirectories. Subdirectories let you organize files into convenient groups. This structure (a top-level directory plus a maximum of seven levels of subdirectories) is called a *directory hierarchy*.

Figure 3–1 shows an example of a directory hierarchy. At the top of the structure is the MFD. The MFD has the name [000000]. It contains entries for user file directories including MALCOLM.DIR, 301300.DIR, and HIGGINS.DIR.

HIGGINS.DIR is the user file directory for HIGGINS. This is the default directory when HIGGINS logs in. HIGGINS.DIR contains entries for two data files, LOGIN.COM and MEMO.LIS. There is also an entry for a directory file, PAYROLL.DIR, which points to the PAYROLL subdirectory.

Figure 3–1 How Directories Are Structured on a Disk



3.4.2 Directory Names

Directory names have the following two possible formats:

- *Named format.* Consists of a top-level directory name that can be followed by a maximum of seven subdirectory names.
- *UIC format.* Contains a two-part octal number that forms a user identification code (UIC). Refers to a user file directory (UFD).

The following sections discuss the rules for using each format.

3.4.2.1 Named Format

A named directory specification has the following format:

[directory-name.subdirectory-name]

The directory name is the name of the top-level directory. It can be followed by up to seven subdirectory names. Default values and wildcard characters can be applied. Observe the following rules when entering a named directory specification:

- A directory name can contain 1 to 39 alphanumeric characters. Characters that are valid for file names are valid for directory names.
- Enclose the directory name in either square brackets ([]) or angle brackets (<>).
- A UFD name can be any group of characters that you request or that the system manager gives you.
- Separate each directory and subdirectory name with a period. For example:

```
$ SET DEFAULT [MAX.TAXES]
$ SET DEFAULT [MAX.TAXES.TESTCASES]
```

- If you specify a directory name that starts with a period, the system places your current default directory before the period. For example, if your current default directory is [POINDEXTER] and you specify [.SQUARE], the system assumes [POINDEXTER.SQUARE].
- You can use certain wildcard characters in a named directory specification. See Section 3.4.3.

3.4.2.2 UIC Format

Almost every DCL command that accepts a file specification can recognize directory names in UIC format. In general, you do not need to use this format unless you are working with RSX systems.

A UIC directory specification has the following format:

[group,member]

where:

- *group* is an octal number that represents a group of users.
- *member* is an octal number that represents a user within the group.

Observe the following rules:

- Ensure the range of each number is 0 to 777 octal.
- Use a comma to separate the group number and the member number.
- Enclose a UIC directory specification in either square brackets ([]) or angle brackets (<>).

For example:

```
[122,1]
```

Directory names in UIC format generally, but not necessarily, correspond to the UIC of the owner of the directory.

You can translate a directory name in UIC format to named format. Zero-fill the group and member fields on the left (if necessary). For example, the named equivalent of the UIC directory specification [122,1] is as follows:

```
[122001]
```

A directory name may not mix UIC format and named format. If you have a directory with a name in UIC format and you want to specify one of its subdirectories, translate the UIC format to named format. For example, [122,1.SUB] is invalid. [122001.SUB] is valid.

For more information on UICs, see Chapter 8.

3.4.3 Searching the Directory Hierarchy

From any point in a directory hierarchy, you can refer to another directory or subdirectory in the structure. You can do this by specifically naming the directory or subdirectory that you want. There are also two wildcard characters available for you to use when referring to directories and subdirectories. They are the hyphen (-) and the ellipsis (...).

Note: You can use the hyphen and ellipsis wildcards in directory specifications that are in named format, but not in UIC format.

This section describes the hyphen and ellipsis wildcards. You can also use the asterisk and percent-sign wildcards in directory specifications. For more information, see Section 3.6.

3.4.3.1 The Ellipsis (...) Wildcard

Use the ellipsis to search downward in the directory hierarchy. To search the current default directory and all the subdirectories below it, use the ellipsis by itself. For example:

```
$ DIRECTORY [...]
```

There are several ways to search specific subdirectories. Starting from a top-level directory, you can search for all subdirectories in the structure that have a certain name. For example:

```
$ DIRECTORY [SMITH...ANALYSIS]
```

This command searches for all subdirectories named [ANALYSIS] under the directory [SMITH]. Some of the subdirectories that would be searched include:

[SMITH.SALES.ANALYSIS]
[SMITH.MARKET.USA.ANALYSIS]

[SMITH.ANALYSIS]

If you begin the directory specification with an ellipsis, the search begins from your current default directory. For example, if your current default directory is [SMITH], the following command searches all subdirectories named [MEMOS] below the default directory [SMITH].

```
$ DIRECTORY [...MEMOS]
```

This command searches subdirectories such as [SMITH.MEMOS] and [SMITH.WORK.MEMOS].

However, if you begin the directory specification with a period, only the subdirectory that is one level lower than the current default directory is searched. For example:

```
$ DIRECTORY [.MEMOS]
```

This command searchs only the [MEMOS] subdirectory that is one level lower than the current default directory. The subdirectory [SMITH.MEMOS] is searched, but [SMITH.WORK.MEMOS] is not.

The next example shows the use of multiple ellipses in a directory specification:

```
$ DIRECTORY [...INVENTORY...]
```

This command searches all subdirectories from the directory named [. . . INVENTORY] below the default directory down to the bottom of the [. . . INVENTORY] hierarchy. Note that in this case [. . . INVENTORY] must be located on a path down from the current directory, and it must be at least one level below the current directory.

The following example also illustrates the use of multiple ellipses in a directory specification:

```
$ DIRECTORY [...INVENTORY...*]
```

In this case, the command searches the subdirectories below the [INVENTORY] subdirectory, but not the [INVENTORY] subdirectory itself.

To search all top-level directories (UFDs) and their subdirectories from wherever you are in the structure, use an asterisk (*) followed by an ellipsis (...). For example:

```
$ DIRECTORY [*...]
```

This specification searches as many as eight levels of directory names (the top-level directory and seven subdirectories), if they exist. It does not search the MFD.

3.4.3.2 The Hyphen (-) Wildcard

The hyphen lets you move up in a directory hierarchy. A single hyphen refers to the directory one level up from the current one. For example, if your current default directory is [JONES.TEST] and you want to go up to [JONES], enter the following command:

```
$ SET DEFAULT [-]
```

To go from [JONES.TEST.BACKUP] to [JONES.TEST.WORK], use the following command:

```
$ SET DEFAULT [-.WORK]
```

The hyphen moves you up one level to [JONES.TEST]. From there, it finds the WORK subdirectory.

You can specify more than one hyphen. For example:

```
$ SET DEFAULT [--.COMPUTE]
```

In this example, you move up two levels in the hierarchy.

If you enter so many hyphens that you point above the MFD, the system displays an error message.

3.4.4 DCL Commands to Use With Directories

The DCL commands listed in Table 3–2 let you move around the directory hierarchy as well as display, create, and delete directories. For more information on any of these commands, see the *VMS DCL Dictionary*.

Table 3–2 DCL Commands to Use With Directories

Command	Function
CREATE/DIRECTORY	Creates a new directory or subdirectory for cataloging files. For example: `$ CREATE/DIRECTORY [JUMBO.LARGE]`

DELETE	Deletes a directory. Note that you can only delete a directory when it is empty. In most cases, before you can delete a directory, you must also change its protection so you (the owner) can delete it. Use the SET PROTECTION command. For example:
	`$ DELETE [.PARTY]*.*;*`
	`$ SET PROTECTION=(O:RWED) PARTY.DIR`
	`$ DELETE PARTY.DIR;1`

DIRECTORY	Displays the files contained in a directory. Qualifiers are available that display a variety of information about a file or group of files. For example: `$ DIRECTORY/SIZE [.PARTY]`
SET DEFAULT	Changes the current default device and directory. For example: `$ SET DEFAULT WORK6:[GAMES]`

Table 3–2 (Cont.) DCL Commands to Use With Directories

Command	Function
SET DIRECTORY	Modifies the characteristics of a directory. You can set a version limit, designate a UIC, or designate an access control list for the files in the directory. For example: `$ SET DIRECTORY/VERSION_LIMIT=6 [ROHBA.MED]`

SHOW DEFAULT	Displays the name of the current default device and directory. For example:
	`$SHOW DEFAULT`
	`  WORK1:[FOGHEAD.RATS]`

3.5 Files

Every file must have a file name or file type to identify it. A file name can also have a version number associated with it. A file name has the following format:

filename.type;version

The following sections describe the definitions and rules for file names, types, and version numbers.

3.5.1 File Names

When you create a file, give it a name that is meaningful to you. A file name can contain 0 to 39 characters. You can use the following characters in a file name:

- A through Z
- a through z
- 0 through 9
- Underscore
- Hyphen
- Dollar sign (reserved for special use by DIGITAL)

Observe the following rules when naming a file:

- You can type any combination of upper- and lowercase letters. However, the system interprets all alphabetic characters as uppercase.
- You cannot begin a file name with a dollar sign, but you can include it within the file name.
- Since a hyphen is the DCL continuation character, you should not end a file name with a hyphen.

3.5.2 File Types

A file type usually identifies the nature of a file. It can contain 0 to 39 characters. The rules for creating file names also apply to file types. In addition, file types must be preceded with a period.

Including a file type is optional. With certain commands, if you omit the file type, the system applies a default value. Table 3–3 lists the default file types used by DCL commands.

Table 3–3 Default File Types

File Type	Contents
ANL	Output file created by the ANALYZE command
BJL	BACKUP journal file
CLD	Command description file
COM	Command procedure file
DAT	Data file
DIF	Output file created by the DIFFERENCES command
DIR	Directory file
DIS	Distribution list file for the MAIL command
DMP	Output file created by the DUMP command
EDT	Startup command file for the EDT Editor
EXE	Image file created by the linker
FDL	File definition language file
HLB	Help text library file
HLP	Input source file for help libraries
INI	Initialization file
JNL	Journal file created by the VMS PATCH Utility
JOU	Journal file created by the EDT Editor
LIS	Listing file created by a language compiler or assembler; default input file type for the PRINT and TYPE commands
LOG	Batch job output file
MAI	Mail message file
MAP	Memory allocation map created by the Linker
MAR	Input source file for the VAX MACRO assembler
MEM	Output file created by DIGITAL Standard Runoff
MLB	Macro library for the VAX MACRO assembler
MSG	Source file that specifies the text of messages
OBJ	Object file created by a language compiler or assembler
OLB	Object module library
OPT	Options file for input to the LINK command
PAR	SYSGEN parameter file
PS	PostScript format file

Table 3–3 (Cont.) Default File Types

File Type	Contents
REGIS	Regis format file
REL	file type for release notes
RELEASE_NOTES	file type for release notes
RNO	Input source file for DIGITAL Standard Runoff
SIXEL	Sixel graphic file
STB	Symbol table file created by the linker
SYS	System image
TEC	TECO indirect command file
TJL	Journal file created by the VAXTPU and ACL editors
TLB	Text library file
TMP	Temporary file
TPU	Command file for the VAXTPU Editor
TXT	Input file for text libraries or MAIL command output
UPD	Update file of changes for a VAX MACRO source program; also input to the SUMSLP Editor

3.5.3 Version Numbers

Version numbers are decimal numbers from 1 to 32,767 that differentiate versions of a file. When you update or modify a file, the system saves both the original file and the modified file. By default, the modified file has the same name and type as the original, but the version number is incremented by one.

Version numbers must be preceded with a semicolon or a period. When the system displays file specifications, it generally displays a semicolon in front of the file version number.

You can refer to versions of a file in a relative manner by specifying a zero or a negative version number. Specifying zero locates the latest (highest numbered) version of the file. Specifying –1 locates the next-most-recent version, –2 the version before that, and so on.

The /VERSION_LIMIT qualifier on the CREATE/DIRECTORY, SET DIRECTORY, and SET FILE commands lets you control the number of versions of a file.

3.5.4 Null File Names and Types

The file name and file type fields can be null. For example, the following are valid file specifications:

.TMP (file name is null)

TEMP. (file type is null)

When you specify a file in a DCL command, be careful to omit the period following a file name if the command uses a default file type. For example, the FORTRAN command uses a default file type of FOR. The following commands produce different results:

```
$ FORTRAN TEMP
$ FORTRAN TEMP.
```

In the first example, the FORTRAN compiler looks for a file named TEMP.FOR because the file type was omitted. In the second example, the compiler looks for a file named TEMP. because a period following the file name indicates a null file type.

3.5.5 Alternate File Names for Magnetic Tapes

In addition to standard file names, the VMS operating system supports an alternate file naming convention for ANSI-labeled magnetic tapes. File names on magnetic tape can have the following format:

"filename".;version

The file name can contain 1 to 17 "a" characters. This set of characters includes numeric characters and uppercase letters, as well as the following:

```
! " % ' ( ) * + , - . / : ; < => ? & _
```

In addition, the asterisk (*) wildcard is allowed in ANSI file names. For more information on wildcards, see Section 3.6.

3.5.6 Specifying a List of Files

If you enter a list of files and do not give a complete file specification for each file in the list, the system applies temporary defaults for the following:

- Node names
- Device names
- Directory names

When not specified by you, the node, device, and directory of a file are determined by the system. The system uses the preceding file specification in the list that included this information. File names and file types can also be defaulted, depending on the command you use. The following examples show how the system applies temporary defaults.

Parameter	How the System Interprets the List
[STATS]A.LIS,B.LIS	[STATS]A.LIS,[STATS]B.LIS
BABE:[STATS]A.LIS, [RUTH]B.LIS,C.LIS	BABE:[STATS]A.LIS,BABE:[RUTH]B.LIS, BABE:[RUTH]C.LIS

To substitute your current default directory for a temporary default, use empty square brackets. The following table shows how the system would interpret a file list if the current default directory were [BETA].

Parameter	How the System Interprets the List
[ALPHA]TEST.DAT,FINAL	[ALPHA]TEST.DAT,[ALPHA]FINAL.DAT
[ALPHA]TEST.DAT,[ ]FINAL	[ALPHA]TEST.DAT,[BETA]FINAL.DAT

If you include a node name in a file that appears in a list, you can override the temporary default by using a double colon.

3.6 Using Wildcards

As mentioned earlier, there are two wildcard characters for use in directory specifications: the hyphen (-) and the ellipsis (...). DCL also provides the following two general purpose wildcard characters that you can use to refer to groups of files:

- Asterisk (*)
- Percent sign (%)

You can use both of these in directory names, file names, and file types. You can use the asterisk in version numbers. You cannot use the percent sign in version numbers or in ANSI magnetic tape file specifications.

The following sections describe the general rules for using wildcard characters in input and output file specifications. Particular uses of wildcard characters in DCL commands vary with the individual commands. For more information on the use of wildcards with a particular command, see the *VMS DCL Dictionary*.

3.6.1 Input File Specifications

The following sections describe how to use wildcard characters in file specifications for input files.

3.6.1.1 The Asterisk (*) Wildcard

Use the * wildcard to match the following:

- An entire field, or a portion of it, in the directory, file name, and file type fields.
- The entire version number field, but not a portion of it.
- An entire ANSI file name, but not a portion of it.

The number of characters that you can match with an asterisk ranges from zero to the maximum size of the field in which the asterisk appears. That is, if the field containing the asterisk permits 39 characters, matches occur on fields that are zero through 39 characters long.

Wildcard characters let you manipulate large numbers of files without naming them individually. For example, the following file specification selects all versions of all files in the [FROGMAN] directory:

```
$ PRINT [FROGMAN]*.*;*
```

You can also limit the files selected to a more specific group:

```
$ TYPE *.DAT;*
```

In this example, only those files in the current default directory with a file type of DAT are displayed.

You can also use the * wildcard in a directory specification:

```
$ DIRECTORY [FROGMAN.*]*.DAT
```

This example selects all files with a file type of DAT that exist in subdirectories one level below [FROGMAN].

Consider another example where wildcard characters appear in the directory specification:

```
$ TYPE [*.*.*]AVERAGE.*;*
```

This file specification selects all versions of all files named AVERAGE, with any file type, that exist in any second-level subdirectory on the current default disk. For example, this file specification selects [A.B.C]AVERAGE.DAT, but not [X.Y]AVERAGE.DAT.

It is also possible to use the * wildcard in directory specifications that are in UIC format. For example, [*,6] indicates all directories with any group number and a member number of 6. The search is limited to directories in UIC format. A directory specification of [*,*] locates all directories in UIC format. To locate all named directories as well as all directories in the UIC format, use [*].

3.6.1.2 The Percent (%) Wildcard

The % wildcard stands for any single character in the position that it occupies. You can use the percent sign in the directory, file name, and file type fields. You cannot, however, use the percent sign in the version number field.

The following example selects all DOC files that have names beginning with CHAP followed by a single character:

```
$ DIRECTORY [FROGMAN]CHAP%.DOC;*
```

All versions of these files in the [FROGMAN] directory are found. Files that might be selected include CHAP2.DOC, CHAP3.DOC, and so forth. CHAP.DOC is not selected because it contains nothing in the percent sign position. Likewise, CHAP24.DOC is not selected because it has more than one character after CHAP. The percent sign replaces only one character position in a field.

You can specify the percent sign as many times as necessary and in combination with other wildcard characters. For example, the following file specification is valid:

```
$ [MA*]INS%%%A*.J*;*
```

The percent sign is not supported for ANSI file names on magnetic tape volumes.

3.6.2 Output File Specifications

The following sections describe how to use wildcards in output file specifications. Only one of the wildcard characters, the asterisk (*), is allowed in the name, type, and version fields for output file specifications. In output directory specifications, two wildcard characters are allowed: the asterisk (*) and the ellipsis (...).

3.6.2.1 Output File Names

You can use the asterisk in the name, type, and version fields in output file specifications. Use an asterisk in an output file specification when you want the output files to match the corresponding field in the input files.

For example, the COPY command generally copies the contents of an input file into a new output file. Using wildcard characters, you can copy large numbers of files without naming them individually. The following example illustrates how DCL interprets asterisks in the output file specifications of the COPY command:

```
$ COPY [FROGMAN]*.*;* USE1:[SAVE.JULY]*.*;*
```

In this case, all the files in the [FROGMAN] directory are copied to another directory on another disk. The file names, types, and version numbers of the copies are the same as those of the original files.

You can also use wildcards to rename files. For example:

```
$ COPY TEST.DAT *.OLD
```

This command copies the highest version of the file TEST.DAT from the current default disk and directory into a file named TEST.OLD.

Note that you cannot use the * wildcard to specify partial name changes. For example, the following command is invalid:

```
$ COPY *1.DAT *2.DAT
```

3.6.2.2 Output Directory Specifications

By including wildcards in output directory specifications, you can do the following:

- Duplicate an entire input directory specification
- Move files from one directory structure into another directory structure at the same or at a different level.

The two wildcard characters you can use in output directory specifications are as follows:

- Asterisk (*)
- Ellipsis (. . .)

Each wildcard character in an output directory specification refers to a corresponding directory level in the input specification. An output specification may contain only wildcards, or it may contain a combination of wildcards and directory names. If directory names are used, they must always precede any wildcards that are included.

Use the asterisk when you want a particular level in the output directory specification to match a level indicated by a wildcard in the input specification. For example:

```
$ BACKUP [JONES.*]*.*;* [SMITH.USER.*]*.*;*
```

This BACKUP command copies all the files from the subdirectories of [JONES] to the corresponding subdirectories of [SMITH.USER]. If the directory [JONES] has a subdirectory [JONES.SUB], then all the files in that subdirectory are copied to the subdirectory [SMITH.USER.SUB]. Notice that the single asterisk in the output directory specification refers to the first subdirectory level in the input directory that contained a wildcard.

Use the ellipsis when you want the output directory specification to follow the same structure downwards as the input directory from the first level that contained a wildcard. For example:

```
$ BACKUP [JONES.SUB...]*.*.* [SMITH...]*.*.*
```

This command copies all the files in [JONES.SUB] to a directory named [SMITH], and all the files in all the subdirectories in [JONES.SUB] to the corresponding subdirectories of [SMITH]. The trailing ellipsis in the output specification lets you move the entire third-level directory structure from the input directory to the second level of the output directory.

For output directory specifications, a trailing asterisk and ellipsis are mutually exclusive whenever they follow a specific directory name. Therefore, output directory specifications such as [USER.* . . .] and [USER . . . *] are invalid. However, [* . . .] is valid, because the asterisk wildcard is used in place of a directory name.

You can move an entire input directory structure to an output directory structure. The two ways to do this are as follows:

```
$ BACKUP DB1:[JONES...]*.*;* DB2:[*]*.*;*
```

or

```
$ BACKUP DB1:[JONES...]*.*;* DB2:[*...]*.*;*
```

These commands let you move all the files in the [JONES] directory structure on DB1 to a [JONES] directory structure on DB2, from the top-level directory down through the entire structure.

You can also use wildcards in input and output directory specifications that are in UIC format. For example:

```
$ BACKUP DB1:[010,*]*.*;* DB2:[*,017]*.*;*
```

In this command, the input group field (010) is substituted for the output group field (*). The output directory becomes DB2:[010,017]. The contents of DB2:[010,017] depend on how many directories there are on DB1 with a group field of 010.

If there is only one directory on DB1 whose group number is 010, all the files from that directory are copied to DB2:[010,017]. If there is more than one directory on DB1 whose group number is 010, all the files in all those directories are copied to DB2:[010,017].

In summary, the following rules apply to using wildcards in output directory specifications:

- An output directory specification composed entirely of wildcards is replaced by the entire input directory structure.
- Trailing asterisks in an output directory are replaced by subdirectories from the input directory, beginning with the first directory in the related input file that contains a wildcard.
- A trailing ellipsis in the output directory is replaced by the structure from the input directory, beginning with the first input directory that contains a wildcard. If, at the current moment, there are no directories with wildcards in the input file specification, then the trailing ellipsis in the output file specification is temporarily ignored for purposes of directory name substitution.
- An asterisk and ellipsis may not both trail in an output directory specification if a directory name is also specified.
- The input and output directory specifications must be in the same format.

3.7 Default Values

When you enter a file specification, you can omit fields and let the system supply default values for these fields. Table 3–4 gives a summary of the defaults applied to each field in a file specification.

Note that the system supplies the defaults described in Table 3–4 for the first input file specification that you enter on a DCL command line. However, when you enter more than one input file specification, the system applies temporary defaults, as described in Section 3.5.6.

Table 3–4 File Specification Defaults

Field	Defaults
node	The system assumes that the device is on the local system.
device	The system uses the device (usually a disk) established at login or by the SET DEFAULT command. Devices are usually identified by using logical names. If a physical device name (ddcu) is used and a controller designation is omitted, the controller designation defaults to A. If a unit number is omitted, the unit number defaults to 0. (The ALLOCATE, MOUNT, and SHOW DEVICES commands, however, treat a device name that does not contain controller or unit numbers as a generic device name.)
directory	The system uses the directory name established at login or by the SET DEFAULT command.
file name	No defaults are applied to the first file name in an input file specification. Most commands apply default output file names based on the file name of an input file.
file type	Various commands apply defaults for file types, based on the standard file type conventions summarized in Table 3–3.
file version	For input files, the system assumes the highest version number. For output files, if no file with the specified file name and file type exists in the current directory, the file is created with a version number of 1. However, if one or more versions do exist, the next highest version number is used.

4 Logical Names

A logical name usually represents a file specification, a device name, or another logical name. Use logical names as a shorthand way of specifying files or directories that you refer to frequently. For example, you might assign a logical name to your default disk and directory. You can also use logical names to do the following:

- Keep your programs and command procedures independent of physical file specifications
- Refer to physical devices

For example, you can assign logical names to tape drives, terminals, and line printers. Also, the system manager assigns logical names to public disk volumes. That way you do not have to be concerned with the physical location of those volumes.

Logical names can be defined by you or by the system. Logical names and their definitions are kept in tables called *logical name tables.* The system provides the following logical name tables:

- Your process table
- The job table for your process
- Your group table
- The system table

When you enter a logical name as part of a command line, the system translates the logical name. It does this by referring to the logical name tables in a certain order. Information about existing logical name tables and the search order are stored in *logical name directory tables.*

With DCL you can create your own logical names and logical name tables. You can also apply special attributes to logical names and define the order in which logical name tables are searched.

The following sections describe logical name tables, logical name directory tables, and logical names provided by the system. Included are all the commands and qualifiers provided by DCL that you need to work with logical names.

4.1 Creating, Displaying, and Deleting Logical Names

You can create your own logical names with either the ASSIGN command or the DEFINE command. This chapter uses the DEFINE command to create logical names. (Note that the syntax for the ASSIGN command differs from the syntax for the DEFINE command. For information on using ASSIGN, see the *VMS DCL Dictionary*.)

The syntax for defining a logical name is as follows:

```
DEFINE logical-name equivalence-name[,...]
```

where:

- *logical-name* is the name you create
- *equivalence-name* is the definition

For example:

```
$ DEFINE CAT WORK1:[SIAMESE]
```

Now you can use CAT to refer to the directory WORK1:[SIAMESE].

Observe the following rules when creating a logical name with the DEFINE command:

- A logical name can be used to form all or part of a file specification.
- A logical name and its equivalence name can each have a maximum of 255 characters. A logical name can contain alphanumeric characters, as well as the underscore (_), dollar sign ($), and hyphen (-).
- The equivalence name must include the punctuation marks (colons, brackets, periods) that would be required if it were part of a file specification. For example, a device name is terminated by a colon, a directory specification is enclosed in square brackets, a file type is preceded by a period.
- You can optionally terminate a logical name with a colon. If you do this, the ASSIGN command removes the colon before placing the logical name in a logical name table. The DEFINE command does not remove the colon before placing the name in a logical name table.

  In general, you should not specify a colon at the end of a logical name when you are creating it. However, if you do specify a colon at the end of a logical name, and if you want to save the colon as part of the logical name, use the DEFINE command. (Note that when you delete a logical name ending with a colon, you need to specify two colons.)

When you use a logical name as *part* of a file specification, the logical name must be the leftmost component of the file specification and must be terminated by a colon (:). When you use a logical name to represent a complete file specification, the terminating colon is not needed.

The following examples illustrate the correct use of the colon with logical names:

Logical Name	Used in a File Specification
DEFINE BOSS DISK$DOC:	DIR BOSS:[RALPH]
DEFINE BOSS DISK$DOC:[RALPH]PLANT.DAT	EDIT BOSS

By default, the DEFINE command places logical names in your process logical name table. To specify a different table, use the /TABLE qualifier. For example:

```
$ DEFINE/TABLE=LNM$JOB CAT WORK1:[SIAMESE]
```

Now the logical name CAT exists in your job logical name table. You can also use the /JOB qualifier to place a logical name in the job table. For example:

```
$ DEFINE/JOB CAT WORK1:[SIAMESE]
```

For more information on logical name tables, see Section 4.2.

4.1.1 Displaying Logical Names

Display the definition of a logical name with either the SHOW TRANSLATION command or the SHOW LOGICAL command.

4.1.1.1 The SHOW TRANSLATION Command

When you enter the SHOW TRANSLATION command, it searches the logical name tables in a certain order for the specified logical name. It displays the first definition it finds as well as the table in which it was found. For example:

```
$ SHOW TRANSLATION CAT
  "CAT" = "WORK1:[SIAMESE]" (LNM$PROCESS_TABLE)
```

This shows that the logical name CAT stands for WORK1:[SIAMESE]. It also shows that the name CAT exists in the process logical name table (LNM$PROCESS_TABLE).

4.1.1.2 The SHOW LOGICAL Command

When you enter the SHOW LOGICAL command, it searches all the logical name tables for the specified logical name. It displays all the definitions it finds as well as the tables in which they were found. For example:

```
$ SHOW LOGICAL CAT
   "CAT" = "WORK1:[SIAMESE]" (LNM$PROCESS_TABLE)
   "CAT" = "WORK1:[SIAMESE]" (LNM$JOB_80F874C0)
```

The system found the logical name CAT in both the process and job logical name tables.

Sometimes the definition of a logical name may include another logical name. The SHOW LOGICAL command keeps searching the logical name tables until all levels of logical names in a definition have been found. This is referred to as *iterative translation*.

When iterative translation is performed, the SHOW LOGICAL command displays multiple lines. Each line has a number that shows the level of translation. For example:

```
$ SHOW LOGICAL MYDISK
  "MYDISK" = "WORK4" (LNM$PROCESS_TABLE)
1 "WORK4" = "$255$DUA17:" (LNM$SYSTEM_TABLE)
```

Level numbers are zero based; that is, 0 is the first level, 1 is the second, and so on. In this example, two translations were performed. The number 1 indicates the second level of translation.

Unless you have redefined the search order, you can display the contents of the process, job, group, and system logical name tables by entering the SHOW LOGICAL command without qualifiers or parameters. For example:

```
$ SHOW LOGICAL
```

This command displays the logical names and their definitions in all four tables.

To display all the entries in a particular logical name table, use the /TABLE qualifier. For example:

```
$ SHOW LOGICAL/TABLE=LNM$GROUP
```

This command displays all the logical names in the group table. You can also use the /GROUP qualifier to display the logical names in the group table. For example:

```
$ SHOW LOGICAL/GROUP
```

The /SYSTEM, /JOB, and /PROCESS qualifiers display the logical names in the system, job, and process tables.

4.1.2 Deleting Logical Names

To delete a logical name, use the DEASSIGN command. For example:

```
$ DEFINE TEST [JONES]
    .
    .
    .
$ DEASSIGN TEST
```

4.2 Logical Name Tables

The system stores logical names in logical name tables. Some logical name tables are available only to your process. Other tables are *shareable*, they are available to other users on the system. Within each table, the system defines some logical names for you. Each table and its system-defined logical names are described in the following sections.

4.2.1 The Process Table

Your process logical name table contains logical names that are available only to your process. Its name is LNM$PROCESS_TABLE. However, you should use the logical name LNM$PROCESS to refer to it. To display the names in your process table, use the following command:

```
$ SHOW LOGICAL/PROCESS
```

By default, the DEFINE and DEASSIGN commands place names in and delete names from your process table.

There is a process logical name table for every process on the system. When you log in, the VMS operating system creates logical names for your process, and places them in your process table. These names are listed in Table 4–1.

Table 4–1 Default Process Logical Names

Logical Name	Description
SYS$COMMAND	The initial file from which DCL reads input. (A file from which DCL reads input is called an *input stream.*) The command interpreter uses SYS$COMMAND to "remember" the original input stream.
SYS$DISK	Default device established at login or changed by the SET DEFAULT command.
SYS$ERROR	The default device or file to which DCL writes error messages generated by warnings, errors, and severe errors.
SYS$INPUT	The default file from which DCL reads input.
SYS$NET	The source process that invokes a target process in DECnet–VAX task-to-task communication. When opened by the target process, SYS$NET represents the logical link over which that process can exchange data with its partner. SYS$NET is defined only during task-to-task communication.
SYS$OUTPUT	The default file to which DCL writes output. (A file to which DCL writes output is called an *output stream.*)
TT	Default device name for terminals.

Note that SYS$INPUT, SYS$OUTPUT, SYS$ERROR, and SYS$COMMAND define files that remain open for the life of the process. They are referred to as *process permanent files.* For more information on process permanent files, see Section 4.9.

Each user on the system is represented by a *job tree.* A job tree is a hierarchy of all your processes and subprocesses, with your main process at the top. Process logical names are recognized by the process they were created in and by any subsequent subprocess. However, process logical names are not recognized by any parent process.

4.2.2 The Job Table

Your job logical name table contains logical names that are available to all processes in your job tree. These logical names are available to you no matter what process or subprocess you are currently in. The name for your job table is LNM$JOB_xxx where xxx is the Job Information Block address (defined by the system) for your job tree. However, you should use the logical name LNM$JOB to refer to your job table.

When you log in, the VMS operating system creates certain logical names and places them in the job logical name table. These names are listed in Table 4–2. In addition, the logical names that are created for mounted disks and tapes and temporary mailboxes are also placed in the job logical name table.

Table 4–2 Default Job Logical Names

Logical Name	Description
SYS$LOGIN	Your default device and directory when you log in.
SYS$LOGIN_DEVICE	Your default device when you log in.
SYS$REM_ID	For jobs initiated through a DECnet network connection, the identification of the process on the remote node from which the job was originated. On VMS operating systems, if proxy logins are enabled, this identification is the process' user name, or, if proxy logins are not enabled, this is the process identification number (PID). For more information about proxy logins, see the *Guide to VMS System Security*. Other operating systems may send other data (terminal numbers, for example) as the remote process identification.
SYS$REM_NODE	For jobs initiated through a DECnet network connection, the name of the remote node from which the job was originated.
SYS$SCRATCH	Default device and directory to which temporary files are written.

There is one job table for each job tree in the system. All job tables are shareable. They are located on the system so that all users may access them. However, to access a job logical name table other than your own, you must redefine LNM$JOB in your process directory logical name table. For more information, see Section 4.3.

4.2.3 The Group Table

The group logical name table contains logical names that are available to all users with the same UIC group number. Its name is LNM$GROUP_xxx where xxx represents your UIC group number. However, you should use the logical name LNM$GROUP to refer to your group table. To create or delete a name in your group table, you need GRPNAM, GRPPRV, or SYSPRV privilege. There is a group logical name table for every group on the system.

4.2.4 The System Table

The system logical name table contains logical names that are available to all users on the system. Its name is LNM$SYSTEM_TABLE. However, you should use the logical name LNM$SYSTEM to refer to it. To create or delete a name in the system table, you must have one of the following:

- A UIC group number between zero and 10
- SYSNAM privilege
- SYSPRV privilege

There is only one system logical name table for the system. It contains the names shown in Table 4–3.

Table 4–3 Default System Logical Names

Logical Name	Description
DBG$INPUT	Default input stream for the VMS Debugger; equated to SYS$INPUT.
DBG$OUTPUT	Default output stream for the VMS Debugger; equated to SYS$OUTPUT.
SYS$COMMON	Device and directory name for the common part of SYS$SYSROOT. (Note that for private system disks, this is identical to SYS$SPECIFIC.)
SYS$ERRORLOG	Device and directory name of error log data files.
SYS$EXAMPLES	Device and directory name of system examples.
SYS$HELP	Device and directory name of system help files.
SYS$INSTRUCTION	Device and directory name of system instruction data files.
SYS$LIBRARY	Device and directory name of system libraries.
SYS$MAINTENANCE	Device and directory name of system maintenance files.
SYS$MANAGER	Device and directory name of system manager files.
SYS$MESSAGE	Device and directory name of system message files.
SYS$NODE	Network node name for the local system if DECnet–VAX is active on the system.
SYS$SHARE	Device and directory name of system shareable images.
SYS$SPECIFIC	Device and directory name for node-specific part of SYS$SYSROOT.
SYS$SYSDEVICE	VMS system disk containing system directories.
SYS$SYSROOT	Device and root directory for system directories. (For a common system disk, this logical name will be defined as a search list that consists of the node-specific device and directory, and the common device and directory. For a noncommon system disk, this logical name will only consist of the node-specific device and directory.)

Table 4–3 (Cont.) Default System Logical Names

Logical Name	Description
SYS$SYSTEM	Device and directory of operating system programs and procedures.
SYS$TEST	Device and directory name of User Environment Test Package (UETP) files.
SYS$UPDATE	Device and directory name of system update files.

4.3 Logical Name Directory Tables

The system provides the following two directory tables to catalog your logical name tables:

- LNM$PROCESS_DIRECTORY catalogs your process tables
- LNM$SYSTEM_DIRECTORY catalogs your shareable tables

Both of these directories contain logical names that translate iteratively to table names. The name of a logical name table must be recorded in one of these directory tables in order for the system to find it.

You can see the relationship of directory tables to logical name tables with the SHOW LOGICAL/STRUCTURE command. For example:

```
$ SHOW LOGICAL/STRUCTURE
(LNM$PROCESS_DIRECTORY)
    (LNM$PROCESS_TABLE)
(LNM$SYSTEM_DIRECTORY)
    (LNM$GROUP_000360)
    (LNM$JOB_806E98E0)
    (LNM$SYSTEM_TABLE)
```

4.3.1 The Process Directory Table

Each process on the system has its own process directory logical name table. When you log in, the VMS operating system places certain logical names in your process directory table. These names are listed in Table 4–4.

Table 4–4 Default Process Directory Logical Names

Logical Name	Description
LNM$GROUP	A logical name that is defined as LNM$GROUP_xxx, where xxx represents your group number. LNM$GROUP_xxx is the logical name table used by your UIC group. (The table LNM$GROUP_xxx is cataloged in the system directory table.) Therefore, LNM$GROUP is a logical name that translates iteratively to the name of your group logical name table.
LNM$JOB	A logical name that is defined as LNM$JOB_xxx, where xxx represents a number unique to your job tree. LNM$JOB_xxx is the logical name table used by your job. (The table LNM$JOB_xxx is cataloged in the system directory table.) Therefore, LNM$JOB is a logical name that translates iteratively to the name of your job logical name table.
LNM$PROCESS	A logical name that is defined as LNM$PROCESS_TABLE. Therefore, LNM$PROCESS is a logical name that translates iteratively to the name of your process logical name table.
LNM$PROCESS_DIRECTORY	The name of your process directory logical name table.
LNM$PROCESS_TABLE	The name of your process logical name table.

4.3.2 The System Directory Table

There is one system directory logical name table. The VMS operating system places certain logical names in the system directory table. These names are listed in Table 4–5.

Table 4–5 Default System Directory Logical Names

Logical Name	Description
LNM$DCL_LOGICAL	A logical name that is defined as LNM$FILE_DEV. This logical name iteratively translates into the list of logical name tables searched and displayed by the SHOW LOGICAL and SHOW TRANSLATION commands and the F$TRNLNM lexical function. By default, these commands search and display the process, job, group, and system logical name tables, in that order.
LNM$DIRECTORIES	A logical name that is defined as LNM$PROCESS_DIRECTORY and LNM$SYSTEM_DIRECTORY.
LNM$FILE_DEV	A logical name that is defined as the list of logical name tables searched by the system when processing a file specification. By default, it is defined as LNM$PROCESS, LNM$JOB, LNM$GROUP, and LNM$SYSTEM. This means that the process, job, group, and system logical name tables are searched, in that order.
LNM$GROUP_xxx	The name of a group logical name table. The characters xxx represent a particular group number. There is an LNM$GROUP_xxx logical name table for each group in the system.
LNM$JOB_xxx	The name of a job logical name table. The characters xxx represent a number unique to this job tree. There is an LNM$JOB_xxx logical name table for each job in the system.
LNM$PERMANENT_MAILBOX	A logical name that is defined as LNM$SYSTEM. (Logical names associated with permanent mailboxes are entered in the logical name table to which the logical name LNM$PERMANENT_MAILBOX iteratively translates.)

Table 4–5 (Cont.) Default System Directory Logical Names

Logical Name	Description
LNM$SYSTEM	A logical name that is defined as LNM$SYSTEM_TABLE. Therefore, LNM$SYSTEM is a logical name that translates iteratively to the name of the system logical name table.
LNN$SYSTEM_DIRECTORY	The name of the system directory logical name table.
LNM$SYSTEM_TABLE	The name of the system logical name table.
LNM$TEMPORARY_MAILBOX	A logical name that is defined as LNM$JOB. (Logical names associated with temporary mailboxes are entered in the logical name table to which the logical name LNM$TEMPORARY_MAILBOX iteratively translates.)

4.4 Logical Name Translation

When the system reads a file specification or device name in a DCL command line, it examines it to see if the leftmost component is a logical name. If the leftmost component ends with a colon, space, comma, or an end-of-line, the system attempts to translate it as a logical name. If the leftmost component ends with any other character, the system does not attempt to translate it as a logical name.

In the following example, the system checks to see if ALPHA is a logical name because ALPHA is the leftmost component of the file specification:

```
$ TYPE ALPHA
```

In the next example, the system checks to see if DISK is a logical name because it is the leftmost component and it ends with a colon (it does not check ALPHA):

```
$ TYPE DISK:ALPHA
```

In the third example, the system does not try to translate [MALCOLM]ALPHA because the leftmost component ends with a square bracket (]):

```
$ TYPE [MALCOLM]ALPHA
```

By default, when the system translates logical names in file specifications, it searches the process, job, group, and system tables, in that order, and uses the first match it finds. The two ways to change this search order are as follows:

- Redefine the logical name LNM$FILE_DEV within the system directory table
- Create a private definition of LNM$FILE_DEV within the process directory table

The system searches the specified tables. For example:

```
$ DEFINE/TABLE=LNM$PROCESS_DIRECTORY -
_$ LNM$FILE_DEV LNM$PROCESS,LNM$SYSTEM
```

In this example, LNM$FILE_DEV is defined so that LNM$JOB and LNM$GROUP are omitted. When a logical name is used in a file specification, the process and system tables are searched but not the job and group tables.

4.4.1 Iterative Translation

Logical name translation can be iterative. This means that after the system translates a logical name, it repeats the process for any logical names it finds contained within the first logical name. For example:

```
$ DEFINE DISK DBA1:
$ DEFINE REPORT DISK:[PAT.STUFF]MAC.SUB
```

In this example, the first DEFINE command equates the logical name DISK to the device name DBA1. The second DEFINE command equates the logical name REPORT to the file specification DISK:[PAT.STUFF]MAC.SUB. When the system translates the logical name REPORT, it finds the equivalence name DISK:[PAT.STUFF]MAC.SUB. It then checks to see if the leftmost component in this file specification ends in a colon, a space, a comma, or an end-of-line. It finds a colon after DISK. The system translates that logical name also. The final translation of the file specification is as follows:

```
DBA1:[PAT.STUFF]MAC.SUB
```

The system limits the number of levels to which it performs logical name translation. The number of levels varies among system facilities, but it is at least nine. If you define more than the system-determined number of levels, or if you create a circular definition, an error occurs when the logical name is used.

4.4.2 Modifying Logical Name Translation

When you create a logical name, you can specify translation attributes that modify the interpretation of an equivalence name. Use the /TRANSLATION_ATTRIBUTES qualifier after the DEFINE command. This is a positional qualifier. Depending on where you place it on a command line, it can apply translation attributes to all equivalence names or only to certain ones.

The following sections describe the two translation attributes that are available: CONCEALED and TERMINAL.

4.4.2.1 Concealing the True Identity of a Logical Name

The CONCEALED attribute causes the logical name of a device, rather than the physical name, to be displayed in system messages. The following example shows how to create a *concealed device name*:

```
$ DEFINE/TRANSLATION_ATTRIBUTES=CONCEALED DISK DMA3:
$ SHOW DEFAULT
  DISK:[ALICE.RATS]
$ SHOW LOGICAL DISK
  "DISK" = "DMA3" (LNM$PROCESS_TABLE)
```

The logical name DISK represents the physical device DMA3. Thus, the SHOW DEFAULT command displays the logical name DISK rather than the actual physical device name, DMA3. The SHOW LOGICAL command reveals the translation of DISK.

You can only use the CONCEALED attribute with logical names that represent physical devices. Using concealed devices lets you write programs, write command procedures, and perform other operations without being concerned about which physical device actually holds the disk or tape. It also lets you use names that are more meaningful than the physical device names.

4.4.2.2 Preventing Iterative Translation

The TERMINAL attribute prevents iterative translation of a logical name. That is, the equivalence name is not examined to see if it is also a logical name. The translation is "terminal" (final, or completed) after the first translation.

4.4.3 How the System Applies Defaults During Logical Name Translation

When the system translates a logical name, it fills in any missing fields in a file specification. It fills them in with the current default device, directory, and version number. When you use a logical name to specify the input file for a command, the command uses the logical name to assign a file specification to the output file as well.

If the equivalence name contains a file name and file type, the output file is given the same file name and file type. If the equivalence name does not contain a file type, a default file type is supplied. The file type supplied depends on the command you are using.

4.4.4 Including a Logical Name in an Input File List

When you use logical names in a list of input files, the equivalence name of each logical name provides a temporary default. For example:

```
$ SET DEFAULT  DBA2:[CASEY]
$ DEFINE MAL DBA1:[MALCOLM]
$ DEFINE HIG [HIGGINS]
$ PRINT ALPHA,MAL:BETA,HIG:GAMMA
```

The PRINT command looks for the following files:

DBA2:[CASEY]ALPHA.LIS
DBA1:[MALCOLM]BETA.LIS

```
DBA1:[HIGGINS]GAMMA.LIS
```

Because a device name was not specified for the logical name HIG, the device name for MAL defines DBA1 as the temporary default device. For more information on temporary defaults, see Section 3.5.6.

4.5 Logical Name Access Modes

The four access modes in the VMS operating system are as follows:

- User-mode (the outermost and least privileged mode)
- Supervisor-mode
- Executive-mode
- Kernel-mode (the innermost and most privileged mode)

When you create a logical name with DCL commands, it has an access mode of user, supervisor, or executive. By default, logical names are created in supervisor mode. There are qualifiers that let you specify user mode or executive mode. For example:

```
$ DEFINE TEST USE1:
$ DEFINE/EXECUTIVE_MODE TEST USE1:
```

The first command places the logical name TEST in the process logical name table in supervisor mode. The second command places the logical name TEST in the process logical name table in executive mode. (You must have SYSNAM privilege to create an executive mode logical name. If you do not have SYSNAM privilege, the name is created in supervisor mode, even though you specified executive mode.) After you enter these commands, two logical names called TEST exist in the process table.

To see the access mode for a logical name, use the SHOW LOGICAL/FULL command. For example:

```
$ SHOW LOGICAL/FULL MAX
  "MAX" [super] = "USE1:[JONES]" (LNM$PROCESS_TABLE)
```

This shows that the logical name MAX was created in supervisor mode.

Logical names created in user mode are temporary. They are deleted when the current image (or the next image, if none is currently active) terminates.

During VMS operations when the integrity of the system could be compromised by incorrect logical names (such as in the activation of privileged images), only executive-mode and kernel-mode logical names are used; supervisor-mode and user-mode names are ignored. DIGITAL therefore recommends that logical names for important system components (public disks and directories, for example) be defined in *executive mode*, using the DCL command DEFINE/SYSTEM/EXECUTIVE. (Only the operating system and privileged programs can create logical names in kernel-mode.)

4.6 Creating Your Own Logical Name Tables

Use the CREATE/NAME_TABLE command to create a new logical name table. The CREATE/NAME_TABLE command creates a logical name table and enters its name in one of the directory logical name tables. (Table names, or logical names that translate iteratively to table names, must always be entered in one of the directory logical name tables.)

You can create logical name tables that are process-private (the default) or shareable. Process-private tables are listed in LNM$PROCESS_DIRECTORY. Shareable tables are listed in LNM$SYSTEM_DIRECTORY.

A name in a directory table can contain 1 to 31 characters. Only alphanumeric characters, the dollar sign ($), and the underscore (_) are valid. For example:

```
$ CREATE/NAME_TABLE NEWTAB
```

This command creates a logical name table called NEWTAB. By default, NEWTAB is entered in LNM$PROCESS_DIRECTORY (the directory table for logical names that are process-private). Use either of the following commands to verify that the table was created:

```
SHOW LOGICAL/TABLE=LNM$PROCESS_DIRECTORY
SHOW LOGICAL/STRUCTURE
```

4.6.1 Shareable Tables

To create a shareable logical name table, use the /PARENT_TABLE qualifier and specify the name of a shareable table. For example:

```
$ CREATE/NAME_TABLE/PARENT_TABLE=LNM$SYSTEM_DIRECTORY NEWTAB
```

The following privilege and access requirements apply to the use of shareable logical name tables:

- To create a shareable logical name table, you must have SYSPRV privilege and ENABLE (E) access to the *parent* table.[1]
- To delete a shareable logical name table, you must have SYSPRV privilege or DELETE (D) access to the table.
- To place a name in or delete a name from a shareable logical name table, you must have WRITE (W) access to the table.
- To read (translate) a name in a shareable logical name table, you must have READ (R) access to the table.

[1] For more information on ENABLE access and logical name tables, see Section 8.2.7.

4.6.2 Choosing a Table for a Logical Name

To place a logical name in a particular table, use the DEFINE/TABLE command. For example:

```
$ DEFINE/TABLE=NEWTAB TESTDIR [JONES.TESTFILES]
```

The logical name TESTDIR is placed in NEWTAB (a user-defined table). However, the next time you refer to TESTDIR in a file specification, the system looks for it in the process, job, group, and system tables only. It does not look in the NEWTAB table. You need to include NEWTAB in the list of tables that the system searches. The two ways to do this are as follows:

- Redefine LNM$FILE_DEV within the system directory table
- Create a private definition of LNM$FILE_DEV within the process directory table

For example:

```
$ DEFINE/TABLE=LNM$PROCESS_DIRECTORY LNM$FILE_DEV -
_$ NEWTAB, LNM$PROCESS, LNM$JOB, LNM$GROUP, LNM$SYSTEM
```

A process-private version of LNM$FILE_DEV is defined to include NEWTAB. During logical name translation, NEWTAB is searched first for the following reasons:

- The process-private version of LNM$FILE_DEV is used before the default system version.
- Within LNM$FILE_DEV, NEWTAB is listed before the process, job, group, and system tables.

Note that you create LNM$FILE_DEV in the process directory table. You cannot change or add things to LNM$SYSTEM_DIRECTORY unless you have SYSNAM or SYSPRV privilege.

4.6.3 Deleting Tables

To delete a logical name table, delete it from the directory table where it is cataloged. For example:

```
$ DEASSIGN/TABLE=LNM$PROCESS_DIRECTORY NEWTAB
```

The table NEWTAB is deleted from the process directory table.

4.6.4 Quotas for Tables

Quotas are used to limit the amount of system resources that a given logical name table can consume. The process, group, and system logical name tables have an infinite quota. By default, when you create a logical name table it too has an unlimited quota.

4.6.4.1 The /QUOTA Qualifier

You can specify a quota to limit the size, in bytes, of a logical name table that you create. For example:

```
$ CREATE/NAME_TABLE/QUOTA=500 ABC
```

This creates a logical name table, ABC, and gives it a quota of 500 bytes. Before a logical name is created, the size of its data structure is checked against the quota remaining for the table. If there is insufficient quota available for the new entry, the system displays an error message.

Once you set the quota for a table, you cannot change it. If the table runs out of room, use the DEASSIGN command to delete old logical names. This frees space for your new logical names.

4.6.4.2 Job Table Quota

The job logical name table is a shareable table. The quota for a job logical name table is established when the table is created. The quota is determined by whichever of the following applies:

- The JTQUOTA value established for the user in the system user authorization file, SYSUAF.DAT (if the first image activated by the process was the VMS system image LOGINOUT).
- The PQL$_JTQUOTA quota list value specified in the call to the Create Process ($CREPRC) system service.
- The /JOB_TABLE_QUOTA qualifier value on the RUN command used to create the detached process.
- The SYSGEN parameter PQL_DJTQUOTA (if none of the preceding conditions applies). The standard default value for this parameter is 1024 bytes; however, the system manager can change it. The SYSGEN utility can be used to display and set the values of the parameters PQL_DJTQUOTA (default job logical name table quota) and PQL_MJTQUOTA (minimum job logical name table quota).

A quota value of 0 for a job logical name table specifies that the quota is, for all practical purposes, unlimited.

4.6.5 Access Modes

When you create a logical name table, it has an access mode. A logical name table can have an access mode of user, supervisor, or executive. By default, logical name tables are created in supervisor mode. You can specify user mode with the /USER_MODE qualifier. If you have SYSNAM privilege, you can specify executive mode with the /EXECUTIVE_MODE qualifier. For more information on access modes, see Section 4.5.

4.6.6 Protection

There are two ways to define the protection of a logical name table:

- Use the /PROTECTION qualifier with the CREATE/NAME_TABLE command. This command lets you set UIC-based protection for a shareable logical name table. For more information, see Chapter 8.
- You can apply access control list (ACL) protection with the ACL Editor or with the SET ACL/OBJECT_TYPE=LOGICAL_NAME_TABLE command. ACLs for system logical name tables are saved, but ACLS for process logical name tables are not. You must reestablish ACLs on process logical name tables every time the system is booted. For more information, see the SET/ACL command in the *VMS DCL Dictionary*.

4.7 Search Lists

A search list is a logical name that has more than one equivalence name. You can use a search list in any place you can use a logical name. For example:

```
$ DEFINE FIFI [JONES.MEMOS],[JONES.WORKFILES]
$ SHOW LOGICAL FIFI

  "FIFI" = "[JONES.MEMOS]" (LNM$PROCESS_TABLE)
           = "[JONES.WORKFILES]"
```

The logical name FIFI is a search list because it has two equivalence names.

When you use a logical name that is a search list, the system translates the logical name a number of times. It uses each equivalence name listed in the definition. For example:

```
$ DIRECTORY FIFI:POEM.DAT

Directory DISK1:[JONES.MEMOS]

POEM.DAT;2      POEM.DAT;1

Total of 2 files.

Directory DISK1:[JONES.WORKFILES]

POEM.DAT;1

Total of 1 file.

Grand total of 2 directories, 3 files.
```

The DIRECTORY command searches the equivalence names [JONES.MEMOS] and [JONES.WORKFILES], in the order they were listed when FIFI was defined. It finds a file named POEM.DAT in each directory. If POEM.DAT exists in only one of the directories, only one directory listing is displayed. If POEM.DAT does not exist in either directory, an error message is displayed indicating that the file was not found.

A search list is not a wildcard. It is a list of places to look. Once a file is found, the search is ended. For example, the following command finds the most recent version of POEM.DAT in the search list defined by FIFI:

```
$ TYPE FIFI:POEM.DAT

DISK1:[JONES.MEMOS]POEM.DAT;2

When in disgrace with fortune and men's eyes
I all alone beweep my outcaste state,
```

However, you can use a search list with a command that accepts wildcards. When you use wildcards, the system forms file specifications using each equivalence name in the search list. The command operates on each file specification that identifies an existing file.

For example, if you specify the TYPE command with a wildcard character in the version field, it finds all versions of POEM.DAT in the search list defined by FIFI.

```
$ TYPE POEM:FILE.DAT;*

DISK1:[JONES.MEMOS]POEM.DAT;2

When in disgrace with fortune and men's eyes
I all alone beweep my outcaste state,
And trouble deaf heaven with


DISK1:[JONES.MEMOS]POEM.DAT;1

When in disgrace with fortune and men's eyes
I all alone beweep my outcaste state,


DISK1:[JONES.WORKFILES]POEM.DAT;1

Let me not to the marriage of true minds
admit after-dinner mints
```

When you use a search list with a command that does not accept wildcards in a file specification, the system forms a file specification using each equivalence name in the search list, until a file specification for an existing file is found. The command performs its function only on the first existing file that is identified by the search list. For example:

```
$ DEFINE SISTER DISK1:[MUFFIN],WORK2:[FRED]
$ EDIT SISTER:AVERAGE.TXT
```

First, the system forms the file specification DISK1:[MUFFIN]AVERAGE.TXT and searches for that file. If the file is found, it is opened so that you can edit it. No other files are subsequently opened. If the first file is not found, the system searches for the file WORK2:[FRED]AVERAGE.TXT. If the file is found, it is opened. If the file is not found, an error message is displayed.

The system displays an error message only after it has used all equivalence names in a search list. Then the system reports an error only on the last file it attempted to find.

Note: The RUN command is an exception. When the RUN command is followed by a search list, the system forms file specifications as described above. However, it then looks for each file specification in the *known file list.* The known file list is a list of all images that are installed on the system. In other words, the system checks to see if any of the files in the list are installed images. It runs the first file in the search list that is an installed image. Then the RUN command terminates.

If none of the file specifications are installed images, the system repeats the process of forming file specifications. This time it looks for each file specification on the disk. It runs the first file it finds there. An error message is displayed if none of the specified files are found in either the known file list or on the disk.

4.7.1 Using Search Lists

When you use a search list in a file specification, the search list is translated as follows:

- If the search list contains only a device, then the original default directory is used
- If the search list contains both a device and a directory, then these are used to construct a complete file specification

For example:

```
$ DEFINE FIFI DISK1:[FRED],DISK2:[GLADYS],DISK3:[MEATBALL.SUB]
$ DIRECTORY FIFI:MEMO.LIS
```

This command displays the following files:

DISK1:[FRED]MEMO.LIS
DISK2:[GLADYS]MEMO.LIS
DISK3:[MEATBALL.SUB]MEMO.LIS

When you specify a search list as the first part of the parameter for the SET DEFAULT command, the system assigns the search list name, untranslated, to SYS$DISK. (SYS$DISK is a logical name that translates to your default disk.) For example:

```
$ SHOW DEFAULT
  DISK2:[MEATBALL.SUB]
$ DEFINE FIFI DISK1:[FRED], DISK2:[GLADYS], DISK3:
$ SET DEFAULT FIFI
$ SHOW DEFAULT
  FIFI:[MEATBALL.SUB]
=   DISK1:[FRED]
=   DISK2:[GLADYS]
=   DISK3:[MEATBALL.SUB]
```

At the beginning of this example, the default disk and directory are DISK2:[MEATBALL.SUB]. Next, a search list is defined. The SET DEFAULT command uses the search list as its parameter. When you enter the SHOW DEFAULT command a second time, the default directory has not changed. However, the search list FIFI is displayed as the default device along with its equivalence names. The SHOW DEFAULT command displays the search list in the order in which the search list is evaluated by the system.

Note: When you specify a search list as the first part of a parameter for the SET DEFAULT command, each equivalence name in the search list must contain a device name.

4.7.2 Search Order for Multiple Search Lists

It is possible to have a file specification that contains more than one search list. When this occurs, each item in the file name search list is used, while the first device name is held constant. After all the items in the file name search list have been used with the first device name, they are used with the second device name. This continues until each device has been used.

The following example shows a file specification that has a search list in the file name and in the device name:

```
$ DEFINE FILE CHAP1.RNO, CHAP2.RNO
$ DEFINE DISK WORK1:[ROSE], WORK2:[THORN]
$ SET DEFAULT DISK
$ DIRECTORY FILE

Directory WORK1:[ROSE]

CHAP1.RNO;2          CHAP2.RNO;1

Total of 2 files.

Directory WORK2:[THORN]

CHAP1.RNO;1          CHAP2.RNO;1

Total of 2 files.

Grand total of 2 directories, 4 files.
```

The directory listing for each file name is given first for WORK1:[ROSE], and second for WORK2:[THORN].

You can also have iterative (nested) search lists, when one name in a search list translates to another search list. If this occurs, the system uses each name in a sublist before continuing on to the next upper level name. For example:

```
$ DEFINE NESTED FRED.DAT, NEW_LIST, RICKY.DAT
$ DEFINE NEW_LIST ETHEL.DAT, LUCY.DAT
```

The search order for the search list NESTED would be as follows:

FRED.DAT
ETHEL.DAT
LUCY.DAT
RICKY.DAT

4.8 Logical Node Names

A logical node name is a special type of logical name that can be used in place of a network node name, or in place of a node name and an access control string. For example:

```
$ DEFINE BOS "BOSTON""ADAMS JOHN"""::"
```

The logical name BOS is equated to a node name, BOSTON, and an access control string where:

- ADAMS is the user name
- JOHN is the password

Use the logical name BOS to avoid typing (and displaying) your user name and password on the terminal screen.

Note: You should not place a DEFINE command that includes a password in a file (LOGIN.COM, for example). If others read the file, they will see the password.

Observe the following rules for using logical node names:

- A logical node name can contain 1 to 15 characters. Its equivalence name must end with a double colon (::). If the equivalence name does not end in a double colon, the logical name is not interpreted as a logical node name. Enclose the equivalence name in quotation marks. Use double quotation marks in the places where you want quotation marks to appear in an access control string.
- A logical node name is translated at the local node. After the logical node name is translated, the rest of the file specification is parsed to determine whether the syntax is valid. However, logical names used as device names are not translated at the local node. Therefore, a file specification can contain a logical node name that is translated at your local node. It can also contain a logical device name that is translated at the remote node. For example:

  ```
  $ DEFINE NYC NEWYRK::
  $ TYPE NYC::DOC:[PERKINS]TERM_PAPER.DAT
  ```

  The logical node name NYC is translated at the local node but the device name (DOC:) is translated at the remote node (NEWYRK). Use a double colon in the DEFINE command to define a logical node name. You must also use a double colon in the TYPE command because NYC is in the node position of the file specification.
- You can specify an access control string with a logical node name when you perform network file operations. This access control string overrides an access control string that is provided in the equivalence name for the logical node name. For example:

  ```
  $ DEFINE BOS "BOSTON""ADAMS JOHN"":::"
  $ TYPE BOS"REVERE PAUL"::RIDE.DAT
  ```

  In this example, the access control string "REVERE PAUL" overrides the access control string provided in the equivalence name.
- Logical node names are translated iteratively. After a logical node name is translated, the new node name becomes a candidate for logical node name translation. (Iterative translation is described in Section 4.4.1.)
- When a logical node name is translated iteratively, the access control information in the logical node name that is first translated overrides subsequent access control information. For example:

  ```
  $ DEFINE TORONTO "TRNTO""TEST RESULTS"":::"
  $ DEFINE TEST1 "TORONTO""TEST 1001"":::DBA1:"
  $ TYPE TEST1:PROC.DAT
  ```

In the previous example, the logical name TEST1 translates to TORONTO"TEST 1001"::DBA1:. TORONTO is a logical node name, so iterative translation occurs. However, the access control string provided by the DEFINE TEST1 logical name assignment overrides the access control string provided in the DEFINE TORONTO logical node name assignment. Therefore, the TYPE command displays the following file:

```
TRNTO"TEST 1001"::DBA1:PROC.DAT
```

- Logical node names that begin with an underscore character are not translated even though the underscore character is not considered part of the node name.

Logical Names for Process-Permanent Files

Process-permanent files are files that can remain open for the life of your process. By default, DCL creates the following process-permanent files for you when you log in:

- SYS$INPUT—default input device or file
- SYS$OUTPUT—default output device or file
- SYS$ERROR—default device or file to which the system writes messages
- SYS$COMMAND—the value of SYS$INPUT when you log in

Table 4–6 shows what these logical names are equated to by default.

Table 4–6 Equivalence Names for Process-Permanent Files

Logical Name	Interactive Mode	Batch Mode	Command Procedure
SYS$COMMAND	Terminal[1]	Disk[2]	Terminal
SYS$INPUT	Terminal	Disk	Disk
SYS$ERROR	Terminal	Log file[3]	Terminal
SYS$OUTPUT	Terminal	Log file	Terminal

[1]Device name of your terminal

[2]Device name of the initial default device

[3]Batch job log file

You can use the logical names for process permanent files as file specifications. The following example shows a FORTRAN command that could be executed in a batch job:

```
$ FORTRAN/OBJECT=WEATHER SYS$INPUT:
```

When this command is executed, the compiler reads the input file from the data lines following the FORTRAN command in the batch job command procedure.

The /OBJECT=WEATHER qualifier provides the compiler with a default file name for the output files: it creates WEATHER.OBJ and WEATHER.LIS. To request that the listing file be written to the batch job log file, you could specify the following:

```
$ FORTRAN/OBJECT=WEATHER/LIST=SYS$OUTPUT   SYS$INPUT:
```

The compiler creates the file WEATHER.OBJ and prints the listing in the batch job log file.

1 Redefining SYS$INPUT

You can redefine SYS$INPUT so that a command procedure reads input from the terminal or another file. For example, to edit a file from a command procedure, include the following lines in the command procedure:

```
$   DEFINE/USER_MODE SYS$INPUT SYS$COMMAND
$   EDIT MYFILE.DAT
.
.
.
```

SYS$INPUT is redefined as SYS$COMMAND so that the editor obtains input from the terminal, rather than from the command procedure file (the default). SYS$COMMAND refers to the terminal, the initial input stream when you logged in. The /USER_MODE qualifier tells the command procedure that SYS$INPUT is redefined only for the duration of the next image. In this case, the next image is the editor. When the editor is finished, SYS$INPUT resumes its default value. In a command procedure, the default value of SYS$INPUT is the command procedure file.

Note that if you redefine SYS$INPUT, DCL ignores your definition. DCL always obtains input from the default input stream. However, images, such as command procedures, can use your definition for SYS$INPUT.

2 Redefining SYS$OUTPUT

You can redefine SYS$OUTPUT to redirect output from your default device to another file. When you redefine SYS$OUTPUT, the system opens a file with the name you specify in the logical name assignment. When you define SYS$OUTPUT, all subsequent output is directed to the new file.

When you redefine SYS$OUTPUT to redirect output from DCL commands that are executed within the command interpreter, the following conditions must be met:

- Create the new logical name definition in supervisor mode and place it in the process logical name table
- Specify only one equivalence name
- Do not specify any attributes for the new logical name

The following example shows a valid definition for SYS$OUTPUT:

```
$ DEFINE SYS$OUTPUT MYFILE.LIS
```

After you enter this command, DCL output is written to MYFILE.LIS, not to the terminal.

When you log in, the system creates two logical names called SYS$OUTPUT. One name is created in executive mode; the other name is created in supervisor mode. You can supersede the supervisor mode logical name by redefining SYS$OUTPUT. If you deassign the supervisor mode name, the system redefines SYS$OUTPUT in supervisor mode, using the executive mode equivalence name. You cannot deassign the executive mode name. For example:

```
$ DEFINE SYS$OUTPUT TEMP.DAT
$ SHOW LOGICAL SYS$OUTPUT
$ SHOW TIME
$ DEASSIGN SYS$OUTPUT
$ TYPE TEMP.DAT

   "SYS$OUTPUT" = "DISK1:" (LNM$PROCESS_TABLE)
  27-JAN-1988 13:26:53
```

First, SYS$OUTPUT is redefined (in supervisor mode) to the file TEMP.DAT. When SYS$OUTPUT is redefined, output from DCL and from images is directed to the file TEMP.DAT. Therefore, the output from the SHOW LOGICAL command (an image) and from the SHOW TIME command (a command that is executed within DCL) is sent to TEMP.DAT. When you deassign SYS$OUTPUT, the system closes the file TEMP.DAT and redefines SYS$OUTPUT (in supervisor mode) to your terminal. Therefore, when you enter the TYPE command, the output is displayed on your terminal.

When you redefine SYS$OUTPUT to a file, the logical name contains only the device portion of the file specification, even though the output is directed to the file you specify. In this example, when SYS$OUTPUT was redefined, the equivalence name contained the device name DISK1:, not the full file specification.

If the system cannot open the file you specify when you redefine SYS$OUTPUT, an error message is displayed.

After you redefine SYS$OUTPUT, most commands direct output to the existing version of the file. However, certain commands create a new version of the file before they write output.

Note that you can redefine SYS$OUTPUT in user mode to redirect output from images.

3 Redefining SYS$ERROR

You can redefine SYS$ERROR to direct error messages to a specified file. However, if you redefine SYS$ERROR so it is different from SYS$OUTPUT (or if you redefine SYS$OUTPUT without also redefining SYS$ERROR), DCL commands send informational, warning, error, and severe error messages to both SYS$ERROR and SYS$OUTPUT. Therefore, you receive these messages twice—once in the file indicated by the definition of SYS$ERROR, and once in the file indicated by SYS$OUTPUT. Success messages are sent only to the file indicated by SYS$OUTPUT.

If you redefine SYS$ERROR and then run an image that references SYS$ERROR, the image sends error messages only to the file indicated by SYS$ERROR—even if SYS$ERROR is different from SYS$OUTPUT. Only DCL commands and images using standard VMS error display mechanisms send error messages to both SYS$ERROR and SYS$OUTPUT when these files are different.

4 Redefining SYS$COMMAND

Although you can redefine SYS$COMMAND, DCL ignores your definition. DCL always uses the default definition for your initial input stream. However, if you execute an image that references SYS$COMMAND, the image can use your new definition.

5 Symbols

A symbol is a name that represents a numeric, character, or logical value. When you use a symbol in a DCL command line, DCL replaces the symbol with its value. Use a symbol in the following ways:

- As a synonym for a DCL command line. Defining a symbol as a command line lets you execute the command by typing only the symbol name. For example, you can create a symbol for a lengthy command line that you type frequently.
- As a variable in a command procedure.
- To refer to data records in command procedures with commands such as READ, WRITE, and INQUIRE.
- To pass a parameter to a command procedure.
- To define a foreign command. Defining a symbol as a foreign command lets you execute a non-DCL image by typing only the symbol name.

This chapter describes the rules for creating symbols.

5.1 Symbol Types

Symbols can be either of the following:

- Local—A local symbol is available to the command level that defined it, any command procedure executed from that command level, and lower command levels.
- Global—A global symbol can be accessed from any command level, regardless of the level at which it was defined.

Local symbols take precedence over global symbols of the same name. Symbols take precedence over identical command names. For example, if you define a symbol with the same name as a DCL command, your definition overrides the command name.

Symbols are stored in the following symbol tables:

- Local symbol table—DCL maintains a local symbol table for your main process and for every command level that you create when you execute a command procedure, use the CALL command, or submit a batch job. A local table is deleted when its associated command level terminates.

  In addition to the local symbols you create, a local symbol table contains eight symbols that are maintained by DCL. These symbols, named P1, P2, and so on through P8, are used for passing parameters to a command procedure. Parameters passed to a command procedure are regarded as character strings. Otherwise P1 through P8 are defined as null character strings (" "). They are stored in the local symbol table.

- Global symbol table—DCL maintains only one global symbol table for the duration of a process. In addition to the global symbols you create, the global symbol table contains the following reserved global symbols:

  $STATUS — The condition code returned by the most recently executed command. $STATUS conforms to the format of a VMS message code.

  $SEVERITY — The severity level of the condition code returned by the most recently executed command. $SEVERITY, which is equal to the three low-order bits of $STATUS, can have the following values:

0	Warning
1	Success
2	Error
3	Information
4	Severe (fatal) error

  $RESTART — Has the value TRUE if a batch job was restarted after it was interrupted by a system crash. Otherwise, $RESTART has the value FALSE.

5.2 Creating Symbols

There are several ways to create a symbol. They are as follows:

- You can define a symbol with an assignment statement.
- You can use the READ and INQUIRE commands. These commands are usually included in command procedures.

This chapter shows how to create symbols with assignment statements. For more information on READ and INQUIRE, see the *Guide to Using VMS Command Procedures* or the *VMS DCL Dictionary*.

An assignment statement has the following format:

```
symbol-name =[=] value
```

The left side of the assignment statement defines the symbol name. The right side of the assignment statement contains a value. A symbol can represent a number, a character string, a lexical function, another symbol, or a combination of these values. Use an equal sign to define a local symbol. A double equal sign defines a global symbol.

Observe the following rules when creating a symbol:

- A symbol name can contain 1 to 255 alphanumeric characters, as well as the underscore (_) and dollar sign ($) characters. The system translates lowercase letters to uppercase.
- Begin a symbol name with a letter, an underscore (_), or a dollar sign ($).

5.2.1 Local Symbols

To create a local symbol with a numeric value, use an equal sign. For example:

```
$ TEST = 15
```

There are two ways to create a local symbol with a character string value. You can use an equal sign and quotes. For example:

```
$ SD = "set default"
```

When you enclose a character string in quotes, lowercase letters are not converted to uppercase; spaces and tab characters are retained.

You can also use a colon and an equal sign. For example:

```
$ SD := set default
```

In this case, character values are converted to uppercase, leading and trailing spaces and tabs are removed; multiple spaces and tabs are compressed to a single space.

A local symbol exists as long as the command level at which it is defined remains active, unless the symbol is specifically deleted.

5.2.2 Global Symbols

To create a global symbol with a numeric value, use two equal signs. For example:

```
$ RESULT == 50
```

There are two ways to create a global symbol with a character string value. You can use two equal signs and quotes (== " "), or a colon and two equal signs (:==). The same rules apply as described in Section 5.2.1.

A global symbol exists for the duration of the process, unless the symbol is specifically deleted.

5.2.3 Symbol Search Order

When the command interpreter determines the value of a symbol, it searches symbol tables in the following order:

1. The local symbol table for the current command level
2. Local symbol tables for each previous command level, searching backwards from the current level
3. The global symbol table

5.2.4 DCL Commands to Use with Symbols

Use the following DCL commands to create, display, and delete symbols.

Table 5–1 DCL Commands to Use with Symbols

Command	Function
DELETE/SYMBOL	Deletes a symbol. By default, the DELETE/SYMBOL command searches for symbols only in the local symbol table. To delete a global symbol, use the /GLOBAL qualifier.
INQUIRE	Reads a value from SYS$COMMAND and assigns it to a symbol.
READ	Reads a record from a file and assigns its contents to a symbol.
SET SYMBOL/SCOPE	You can mask global or local symbols at the specified command level.
SHOW SYMBOL	Displays the value of the specified symbol. By default, the SHOW SYMBOL command searches the local symbol tables and then the global symbol table to locate a specified symbol name.

5.2.5 Abbreviating Symbol Names

You can use abbreviated forms of symbols if you define them with the asterisk. The following example shows how to create a local symbol that can be abbreviated:

```
$ M*AIL = "MAIL"
```

The VMS Mail Utility is executed whenever the following versions of the symbolic name are used:

```
$ M
$ MA
$ MAI
$ MAIL
```

Generally, you can use abbreviated symbol definitions in any situation that allows a symbol to be used. However, there are some restrictions:

- You cannot abbreviate symbols that involve substring replacement.
- When you define a symbol that includes an asterisk, existing symbols may be deleted. If an existing symbol exactly matches the new symbol at or past the asterisk, the new symbol replaces the existing symbol.
- If you define a symbol with an asterisk, you cannot define another symbol whose name partly matches the existing symbol at or past the asterisk.

5.3 Values Used in Symbols

A symbol can be defined as a character string, a number, a lexical function, another symbol, or a combination of these values. The following sections describe these values.

5.3.1 Character Strings

A character string can contain any characters that can be printed. You can define a character string by enclosing it in quotes. In this way, alphabetic case and spaces are preserved when the symbol assignment is made. Note the following:

- To include quotation marks within a string, type two consecutive quotation marks. For example:

```
$ PUP = "Type ""YES"" to proceed"
```

- To continue a character string over two lines, use a plus sign (for string concatenation) and a hyphen (for continuation). For example:

```
$ HEAD = "MONTHLY REPORT --" + -
_$ " DECEMBER 1988"
```

You can also define a character string with a colon and one or two equal signs (:= or :==). In this way, all alphabetic characters are converted to uppercase, and spaces are compressed. To continue a character string over two lines in this format, use a single hyphen.

You can also concatenate several symbols to create a long character string. Do not place quotation marks around symbols when you use them in expressions. For example:

```
$ DOG1 = "Ralph, "
$ DOG2 = "ruthless pup"
$ DOG3 = DOG1 + DOG2
$ SHOW SYMBOL DOG3
   DOG3 = "Ralph, ruthless pup"
```

5.3.2 Numbers

A number can have the following values:

- Decimal—the ASCII characters 0 through 9
- Hexadecimal—the ASCII characters 0 through 9 and A through F
- Octal—the ASCII characters 0 through 7

You can specify a number as follows:

- Positive number—Specify a positive number by typing the appropriate digits. For example:

```
$ COUNT = 13
```

- Negative number—Precede a negative number with a minus sign. For example:

```
$ COUNT = -13
```

- Radix—Specify a number in a radix other than decimal by preceding the number with a %X for hexadecimal numbers and %O for octal numbers. There cannot be any blanks between a radix operator and a value. For example:

```
$ COUNT = %XD
$ SHOW SYMBOL COUNT
   COUNT = 13   Hex = 0000000D   Octal = 00000000015
```

  To specify a negative number in a radix, precede the percent sign (%) with a minus sign (-). For example:

```
$ NEG = -%XD
$ SHOW SYMBOL NEG
   NEG = -13   Hex = FFFFFFF3   Octal = 37777777763
```

  When you specify a value in either hexadecimal or octal, the command interpreter converts the value to a decimal integer.

5.3.3 Lexical Functions

Lexical functions return information about a requested item. Type the name of the lexical function (which always begins with F$) and its argument list. Use the following syntax:

F$function-name(args[,...])

Use lexical functions the same way you would use character strings, integers, and symbols. The following example uses the F$LENGTH function. F$LENGTH returns an integer that specifies the length of the string. The returned value is assigned to the symbol LEN.

```
$ LEN = F$LENGTH("The cat jumped over the moon.")
$ SHOW SYMBOL LEN
  LEN = 29   Hex = 0000001D  Octal = 000035
```

Observe the following rules:

- Do not enclose lexical functions in quotation marks.
- Enclose the argument list in parentheses.
- If an argument contains a character string, enclose the character string in quotation marks.
- If an argument contains an integer, a symbol, or another lexical function, do not enclose these values in quotation marks.

For a complete description of each lexical function, its required arguments, and its return values, see the *VMS DCL Dictionary*.

5.3.4 Another Symbol

After a symbol is defined, it can be used as a value for another symbol. It can be interpreted as a character string or a number, depending on the context in which it is used. For example, suppose a symbol, COUNT, is assigned the integer value 3:

```
$ COUNT = 3
```

Then the value of COUNT can be used in other assignment statements as shown in the following examples.

```
$ TOTAL = COUNT + 1
```

The value of COUNT is added to 1. The result, 4, is equated to the symbol TOTAL.

```
$ SHOW SYMBOL TOTAL
  TOTAL = 4    Hex = 00000004  Octal = 00000000004
```

You can include the symbol COUNT in a string assignment statement. For example:

```
$ BARK := P'COUNT'
```

COUNT is converted to a string value and appended to the character P. BARK now has the value P3.

To include a symbol in a string assignment, follow these rules:

- Use either a colon and an equal sign (:=) or a colon and two equal signs (:==).
- Enclose the symbol in apostrophes. Otherwise DCL will not recognize it as a symbol.

If you define a null character string value for a symbol, that symbol has a value of 0 when it is used in an arithmetic context. For example:

```
$ A = ""
$ B = 2
$ C = A + B
$ SHOW SYMBOL C
   C = 2  Hex = 00000002  Octal = 00000000002
```

5.3.5 Combination of Values

A symbol can be defined as a combination of values, called an *expression*. Within an expression each value is regarded as an *operand*. An operand is connected to another operand by an *operator*. Operators specify the operations to be performed. For more information on how DCL handles expressions and operators, see Chapter 6.

5.4 Foreign Commands

You may have a non-DCL image that you run frequently. Rather than typing the entire file specification of the image every time you want to run it, you can equate it to a symbol. This way you can run the image by typing the symbol. A symbol that runs an image is referred to as a *foreign command.* A foreign command is an image that is not recognized by the command interpreter as a DCL command.

Use either of the following formats to define a foreign command:

```
$ symbol-name :=[=] $image-file-spec
$ symbol-name =[=] "$image-file-spec"
```

where:

- *symbol-name* is the name you want to use to run the image.
- *$image-file-spec* is the file specification of the image. The dollar sign ($) preceding image-file-spec is required. The default device and directory name is SYS$SYSTEM, the default file type is EXE, and the default file version number is the highest version.

The following example defines the symbol PROCESS as a foreign command (note that the file specification begins with a $):

```
$ PROCESS := $DB1:[SARAH.PROG]CREPROCES
```

The request to run the image is implied in the symbol definition. In a command line, PROCESS could be followed by a parameter. In the following example, the file specification RAT.DAT is a parameter that is passed to the image defined by PROCESS:

```
$ PROCESS RAT.DAT
```

The command interpreter looks for symbols enclosed by apostrophes and translates them. Thus, if you use symbols or lexical functions preceded by apostrophes to specify parameters, symbol substitution occurs. Otherwise the command interpreter does not parse the line. The image must obtain the parameter and perform any parsing or evaluation of the command line.

You can use an abbreviated form of a foreign command if you use an asterisk (*) when you define the foreign command.

An alternative to using a foreign command is to define new commands with the Command Definition Utility. See the *VMS Command Definition Utility Manual* for more information.

6 More on Expressions

An expression is a combination of values. A value is referred to as an operand. Each value, or operand, is connected to another value by an operator. For example:

```
$ BARK = 1 + 2 + 3
```

The symbol BARK is equated to an expression that adds three numbers. In this case, the operands are 1, 2, and 3. The operator is the plus sign (+).

Expressions can evaluate to either character strings or integers. The following sections describe the rules DCL uses to evaluate expressions.

6.1 Character String Expressions

A character string expression can contain character strings, lexical functions that evaluate to character strings, or symbols that have string values. It can also contain string operands that are connected by operators. For example, the following symbols are equated to character string expressions:

```
$ TEMP = "CAT"
$ TOPIC = "THE" + TEMP
$ COUNT = F$STRING(65)
$ TOTAL = "NUMBER OF FILES = " + F$STRING(16)
$ SUBTOTAL = TOTAL
```

Some of these expressions contain a single value that is a character string, another symbol, or a lexical function. However, some string expressions contain character strings, symbols, and lexical functions that are connected with string operators. For example, the symbol TOPIC contains a character string ("THE ") and another symbol (TEMP):

```
$ TEMP = "CAT"
$ TOPIC = "THE " + TEMP
$ SHOW SYMBOL TOPIC
   TOPIC = "THE CAT"
```

For string operations to occur, both operands must have character string values. When you use a character string in an expression, place quotation marks around it. However, when you use a symbol in an expression, do not use quotation marks.

More on Expressions

6.1 Character String Expressions

6.1.1 String Operations

The result of a string operation is a character string value. Use the following operators to perform string operations.

+ String concatentaion. The plus sign concatenates two character strings to form a single character string.

– String reduction. The minus sign subtracts one character string from another.

Observe the following rules:

- In order for string concatenation or reduction to occur, all operands must be character string expressions. Otherwise, any string will be converted to an integer and the result will be an integer.
- If the string following the minus sign in a string reduction operation occurs more than once in the preceding string, only the first occurrence is removed.

Following are some examples of string operations:

Expression	Result
A = "MYFILE" + ".MEM"	"MYFILE.MEM"
B = "FILENAME.MEM" – "FILE"	"NAME.MEM"
C = "LISTING.LIS" – "LIS"	"TING.LIS"
D = "MIS" + C	"MISTING.LIS"

6.1.2 String Comparisons

The result of a string comparison is either true (1) or false (0). It is based on the binary values of the ASCII characters in the string. (The ASCII characters and their hexadecimal values are listed in Appendix B.) Use the following operators in string comparisons:

.EQS. Equal to

.GES. Greater than or equal to

.GTS. Greater than

.LES. Less than or equal to

.LTS. Less than

.NES. Not equal to

Observe the following rules:

- The comparison is on a character-by-character basis. The comparison terminates as soon as two characters do not match.
- If the result of a comparison is true, the expression is given a value of 1. If the comparison is false, the expression is given a value of 0.
- If one string is longer than the other, the shorter string is padded on the right with nulls (an ASCII value of %X00) before the comparison is made. Null has a lower numeric value than any of the alphanumeric characters.

- Lowercase letters have higher numeric values than uppercase letters.
- Operands in string comparisons are string expressions. If you specify an integer value as an operand, it is converted to a string before the comparison is performed. For information on how DCL converts integers to strings, see Section 6.4.2.
- If you do not enclose a character string in quotation marks, the command interpreter assumes the string is a symbol name. If the symbol is not defined, an error message is displayed.

Following are some examples of string comparisons:

Expression	Result	Explanation
"MAYBE".LTS."maybe"	1 (true)	The expression is true because the ASCII value of "M" is less than "m."
"ABCD".LTS."EFG"	1 (true)	The expression is true because the ASCII value of "A" is less than "E."
"YES".GTS."YESS"	0 (false)	The expression is false because the ASCII value of a null character is less than the ASCII value of "S".
"AAB".GTS. "AAA"	1 (true)	The expression is true because the ASCII value of "B" is greater than "A".
"TRUE".EQS.1	0 (false)	DCL converts the integer 1 to the string "1" before comparing the ASCII value of "T" to the ASCII value of "1".
"FALSE".EQS.0	0 (false)	DCL converts the integer 0 to the string "0" before making the comparison.
"123".EQS.123	1 (true)	DCL converts the integer 123 to the string "123" before making the comparison.

6.1.3 Replacing Substrings

You can replace a part of a character string with another character string. The assignment statement has the following format:

```
symbol-name[offset,size]:= replacement-string
```

or

```
symbol-name[offset,size]:== replacement-string
```

where:

- *offset* is an integer that indicates the position of the replacement-string relative to the first character in the original string. An offset of 0 means the first character in the symbol, an offset of 1 means the second character, and so on.
- *size* is an integer that indicates the length of the replacement-string.

Observe the following rules:

- The square brackets are required notation. No spaces are allowed between the symbol name and the left bracket.
- Integer values can be in the range of 0 through 768.
- The replacement-string must be a character string.

For example:

```
$ A := PACKRAT
$ A[0,4] := MUSK
$ SHOW SYMBOL A
   A = "MUSKRAT"
```

The first assignment statement gives the symbol A the value PACKRAT. The second statement specifies that MUSK replace the first four characters in the value of A. The result is that the value of A becomes MUSKRAT.

The symbol name you specify can be undefined initially. The assignment statement creates the symbol name and, if necessary, provides leading or trailing spaces in the symbol value. For example:

```
$ B[4,3]:= RAT
```

If the symbol B does not have a previous value, it is given a value of four leading spaces followed by RAT. This format creates a blank line of any length. The following example gives the symbol LINE a value of 80 blank spaces:

```
$ LINE[0,80]:= " "
```

Lining up records in columns makes a list easier to read and sort. You can use this format to specify how you want data to be stored. For example:

```
$ DATA[0,15] := 'NAME'
$ DATA[17,1] := 'GRADE'
```

The first statement fills in the first 15 columns of DATA with whatever value NAME has. The second statement fills in column 18 with whatever value GRADE has. Columns 16 and 17 contain blanks. To see how this would be useful, compare the following command procedures and their output. The first command procedure does not use an assignment statement to format the output.

```
$ !FIRST.COM
$ OPEN/WRITE RECORD LOG.DAT
$ LABEL:
$ INQUIRE NAME "Name"
$ INQUIRE GRADE "Grade"
$ WRITE RECORD NAME,GRADE
$ INQUIRE MORE "More (y/n)"
$ IF MORE THEN GOTO LABEL
$ CLOSE RECORD
```

The output of FIRST.COM would look like the following:

```
BOBB
JOA
MEREDITHC
RALPHF
```

The second version uses the assignment statements discussed above to format the output.

```
$ !SECOND.COM
$ OPEN/WRITE RECORD LOG.DAT
$ LABEL:
$ INQUIRE NAME "Name"
$ INQUIRE GRADE "Grade"
$ DATA[0,15] := 'NAME'
$ DATA[17,1] := 'GRADE'
$ WRITE RECORD DATA
$ INQUIRE MORE "More (y/n)"
$ IF MORE THEN GOTO LABEL
$ CLOSE RECORD
```

The output of SECOND.COM would look like the following:

```
BOB             B
JO              A
MEREDITH        C
RALPH           F
```

Figure 6–1 illustrates some applications of string substitutions using offsets.

More on Expressions

6.1 Character String Expressions

Figure 6–1 Replacing Character Strings in Assignment Statements

Interactive Assignment	Resulting Symbol Value	Comments
`$ FILENAME:=MYFILE.DAT`	MYFILE.DAT	The result is the initial value of symbol FILENAME
`$ FILENAME[0,2]:=TR`	TRFILE.DAT	Two characters starting at offset 0 are overlaid
`$ FILENAME[2,4]:=TESTING.LIS`	TRTEST.DAT	When the string value is longer than the character count the value is truncated to the count
`$ FILENAME[10,2]:=;1`	TRTEST.DAT;1	When the length of the string is equal to the value of the offset, the string is appended to the current value
`$ COMMAND:=TYPE`	TYPE	This is the initial value of the symbol command
`$ COMMAND[5,13]:='FILENAME`	TYPE TRTEST.DAT;1	Appends a space and the current value of FILENAME to the TYPE command verb.

ZK-818-82

6.2 Numeric Expressions

A numeric expression can contain integers, lexical functions that evaluate to integers, or symbols that have integer values. It can also contain integer operands that are connected by arithmetic, logical, and comparison operators. For example, the following symbols are equated to numeric expressions:

```
$ COUNT = 1
$ VALUE = %X1C
$ SUM = 1 + 7 - 4/3 + 10
```

6.2.1 Numeric Operations

The result of a numeric operation is an integer. Use the following operators:

+	Addition
–	Subtraction
+	Unary plus (indicates a positive number)
–	Unary negate (indicates a negative number)
*	Multiplication
/	Division (integer quotient)

Observe the following rules:

- Operands in numeric operations are numeric expressions. If you specify a string value as an operand, it is converted to an integer value before the operation is performed. For information on how DCL converts a string to an integer, see Section 6.4.1.
- All nondecimal values (specified by radix operators) are converted to integer values before the operation is performed.
- All arithmetic is integer arithmetic. Fractional values are truncated. For example, 5 divided by 2 equals 2.

Following are some examples of numeric operations:

Expression	Result
A = 5 + 4	A = 9
B = –5 + 4	B = –1
C = 6/3	C = 2
D = 6 / 4	D = 1
E = %X10 + 5	E = 21

6.2.2 Numeric Comparisons

The result of a numeric comparison is either true (1) or false (0). Use the following operators in numeric comparisons:

.EQ.	Equal to
.GE.	Greater than or equal to
.GT.	Greater than
.LE.	Less than or equal to
.LT.	Less than
.NE.	Not equal to

Observe the following rules:

- If the result of a comparison is true, the expression is given a value of 1. If the result of the comparison is false, the expression is given a value of 0.

- Operands in numeric comparisons are numeric expressions. If you specify a character string value as an operand, it is converted to an integer value before the comparison is performed. For information on how DCL converts a string to an integer, see Section 6.4.1.

Following are some examples of numeric comparisons:

Expression	Value of Expression
1.LE.2	1 (true)
1.GT.2	0 (false)
1 + 3 .EQ. 2 + 5	0 (false)
"TRUE".EQ.1	1 (true)
"FALSE".EQ.0	1 (true)
"123".EQ.123	1 (true)

6.2.3 Logical Operations

A logical operation affects all the bits in the number being acted upon. The result of a logical operation is an integer. Operands for logical operations are integer expressions. If you specify a character string value as an operand, it is converted to an integer value before the operation is performed. Note the following:

- If the first character is T, t, Y, or y, a character string has a logical value of true (1).
- If the first character is not T, t, Y, or y, a character string has a logical value of false (0).
- If an integer is odd (the low-order bit is 1), it has a logical value of true (1).
- If an integer is even (the low-order bit is 0), it has a logical value of false (0).

Use the following operators:

- .NOT.—Reverses the bit configuration of a logical value. A true value becomes false and a false value becomes true. In the following example, the value of STATUS (-2) is even, or false:

```
$ SHOW SYMBOL STATUS
 STATUS = 1  Hex = 00000001  Octal = 00000000001
$ STATUS = .NOT. STATUS
$ SHOW SYMBOL STATUS
 STATUS = -2  HEX = FFFFFFFE  Octal = 37777777776
```

- .AND.—Combines two logical values as follows:

Bit Level	Entity Level
1 .AND. 1 = 1	true .AND. true = true
1 .AND. 0 = 0	true .AND. false = false
0 .AND. 1 = 0	false .AND. true = false
0 .AND. 0 = 0	false .AND. false = false

For example:

```
$ STAT1 = "TRUE"
$ STAT2 = "FALSE"
$ STATUS = STAT1 .AND. STAT2
$ SHOW SYMBOL STATUS
 STATUS = 0   Hex = 00000000   Octal = 000000
```

- .OR.—Combines two logical values as follows:

Bit Level	Entity Level
1 .OR. 1 = 1	true .OR. true = true
1 .OR. 0 = 1	true .OR. false = true
0 .OR. 1 = 1	false .OR. true = true
0 .OR. 0 = 0	false .OR. false = false

For example:

```
$ STAT1 = "TRUE"
$ STAT2 = "FALSE"
$ STATUS = STAT1 .OR. STAT2
$ SHOW SYMBOL STATUS
 STATUS = 1   Hex = 00000001   Octal = 000001
```

6.2.4 Numeric Overlays

A special format of the assignment statement can be used to perform binary overlays of the current symbol value. This format is as follows:

$ symbol-name[bit-position,size] = replacement-expression

or

$ symbol-name[bit-position,size] == replacement-expression

where:

- *bit-position* is an integer that indicates the location relative to bit 0 at which the overlay is to occur.
- *size* is an integer that indicates the number of bits to be overlaid.

Observe the following rules:

- The square brackets are required notation. No spaces are allowed between the symbol name and the left bracket.
- Literal values are assumed to be decimal.

- The maximum length for both *bit-position* and *size* is 32 bits.
- The *replacement-expression* must be a numeric expression.
- When *symbol-name* is either undefined or defined as a string, the result of the overlay is a string. Otherwise the result is an integer.

The following example defines the symbol BELL as the value 7. The low-order byte of BELL has the binary value 00000111. By changing the 0 at offset 5 to 1 (beginning with 0, count bits from right to left), you create the binary value 00100111 (decimal value 39).

```
$ BELL = 7
$ BELL[5,1] = 1
$ SHOW SYMBOL BELL
 BELL = 39   Hex = 00000027   Octal = 00000000047
```

6.3 Order of Operations

An expression can contain any number of operations and comparisons. When an expression contains more than one operation, the operations are performed in a certain order. Table 6–1 lists the operations you can use and the order in which they are performed by DCL. Operators at the top of the table are performed before operators at the bottom. Operators with the same precedence are performed from left to right, as they appear in the command line.

Table 6–1 Order of Operations

Operator	Precedence	Description
+	7	Indicates a positive number
–	7	Indicates a negative number
*	6	Multiplies two numbers
/	6	Divides two numbers
+	5	Adds two numbers or concatenates two character strings
–	5	Subtracts two numbers or reduces two strings
.EQS.	4	Tests if two character strings are equal
.GES.	4	Tests if first string is greater than or equal to the second
.GTS.	4	Tests if first string is greater than the second
.LES.	4	Tests if first string is less than or equal to the second
.LTS.	4	Tests if first string is less than the second
.NES.	4	Tests if two strings are not equal
.EQ.	4	Tests if two numbers are equal
.GE.	4	Tests if first number is greater than or equal to the second
.GT.	4	Tests if first number is greater than the second

Table 6–1 (Cont.) Order of Operations

Operator	Precedence	Description
.LE.	4	Tests if first number is less than or equal to the second
.LT.	4	Tests if first number is less than the second
.NE.	4	Tests if two numbers are not equal
.NOT.	3	Logically negates a number
.AND.	2	Combines two numbers with a logical AND
.OR.	1	Combines two numbers with a logical OR

Observe the following rules:

- Begin and end logical and comparison operators with a period (.). Intervening blanks are not allowed.
- You can type any number of blanks or tabs between operators and operands. For example, the following expressions are equivalent:

```
A.EQS.B
A .EQS. B
```

- Each operator (except .NOT. and the unary plus or minus signs) must have an operand on each side. Note that the unary plus sign can be the initial character in an assignment statement, but not in a WRITE statement.
- Use parentheses to override the order of operations. Items within parentheses are calculated first.

Following are some examples:

Expression	Result
A = 5 + 10 / 2	A = 10
B = 5 * 3 - 4 * 6 / 2	B = 3
C = 5 * (6 - 4) - 8 / (2 - 1)	C = 2

6.4 Value Type Conversion

All the operands in an expression must be of the same value type (number or character string) before DCL can evaluate the expression. When an expression contains both number and string operands, DCL either converts all strings to integers, or all integers to strings.

For example, if you include an integer in a string comparison, DCL converts the integer to a string. In other expressions that contain both integer and string values, DCL converts the strings to integers.

In addition, the following lexical functions let you determine or change the value of an expression:

- F$TYPE—Determines the current value type of a symbol

- F$INTEGER—Converts a string expression to an integer value
- F$STRING—Converts an integer expression to a string value

The following sections describe how DCL converts an integer to a string and vice versa.

6.4.1 String to Integer Conversion

Character strings are converted to integers in the following ways:

- Strings containing numbers are converted to their integer values. For example, the string "45" is converted to the integer 45.
- If a character string begins with T, t, Y, or y, it is converted to the integer 1.
- If a string begins with any other letter, it is converted to the integer 0.

The following examples show how strings are converted to integer values:

String	—>	Resulting Integer
"123"	—>	123
"12XY"	—>	0 (False)
"Test"	—>	1 (True)
"hello"	—>	0 (False)

6.4.2 Integer to String Conversion

When integers are converted to character strings, the resulting string contains numbers that correspond to the integer value. The following examples show how integers are converted to string values:

Integer	—>	Resulting String
123	—>	"123"
1	—>	"1"
0	—>	"0"

6.4.3 How DCL Evaluates an Expression

An expression has either an integer or a string value, depending on the types of values and the operators used. Table 6–2 summarizes how DCL evaluates expressions. The first column lists the different operands and operators that an expression might contain. The second column tells, for each case, what the entire expression is equated to. Within the table *any value* stands for a string or an integer.

Table 6–2 Determining the Value of an Expression

Expression	Resulting Value Type
Integer value	Integer
String value	String
Integer lexical function	Integer
String lexical function	String
Integer symbol	Integer
String symbol	String
+, –, or .NOT. any value	Integer
Any value .AND. or .OR. any value	Integer
String + or - string	String
Integer + or - any value	Integer
Any value + or - integer	Integer
Any value * or / any value	Integer
Any value (string comparison) any value	Integer
Any value (numeric comparison) any value	Integer

7 Symbol Substitution

The DCL command interpreter looks for certain cues that indicate when an item in a command line is a symbol. When it finds an item that seems to be a symbol, the command interpreter replaces the symbol with its current value. Replacing a symbol with its current value is referred to as *symbol substitution.*

Symbol substitution happens automatically in certain situations. You can also force DCL to recognize an item as a symbol with *substitution operators.* This chapter describes how the command interpreter performs symbol substitution in both cases.

7.1 Automatic Symbol Substitution

The command interpreter assumes, in certain contexts, that a string beginning with an alphabetic character is a symbol name or a lexical function. DCL automatically tries to perform symbol substitution on character strings in the following contexts:

- On the right side of an = or == assignment statement. For example:

  ```
  $ TOTAL = COUNT + 1
  ```

  COUNT is automatically recognized and evaluated as a symbol.

- In a lexical function. For example:

  ```
  $ QUERY = "Haven't we met before?"
  $ LEN = F$LENGTH(QUERY) + 5
  $ SHOW SYMBOL LEN
    LEN = 27   Hex = 0000001B  Octal = 000033
  ```

  In the second line, the symbol QUERY is automatically evaluated when it is used with the F$LENGTH function. Also, the F$LENGTH function is automatically evaluated because it is on the right side of an assignment statement.

- In a DEPOSIT, EXAMINE, or IF command. For example:

  ```
  $ IF A .EQ. B THEN WRITE SYS$OUTPUT "DONE"
  ```

  The IF command assumes that both A and B are symbol names and uses their current values.

- At the beginning of a line when the string is not followed by an equal sign or a colon. For example:

  ```
  $ PDEL = "DELETE SYS$PRINT/ENTRY="
  $ PDEL 181
  ```

  In the second line, the command interpreter automatically replaces PDEL with its current value and executes the resulting command.

- In the brackets on the left side of an assignment statement when you are performing substring replacement or numeric overlays.

In any of these contexts, the command interpreter assumes that any character string beginning with an alphabetic character is a symbol name and that any string beginning with a number or with the radix operator (%) is a literal numeric value.

7.2 Substitution Operators

You can use a substitution operator to request symbol substitution in places where DCL does not usually perform it. DCL accepts two substitution operators:

- Apostrophe (')
- Ampersand (&)

The difference between these two operators is the time when the substitution occurs. Symbols preceded by apostrophes are substituted during the first phase of command processing. Symbols preceded by ampersands are substituted during the second phase of command processing. For more information on the phases of command processing, see Section 7.3.

7.2.1 The Apostrophe (')

The apostrophe is the normal substitution operator. Use it to request symbol substitution when you use a symbol in place of a command parameter or qualifier. For example:

```
$ LIT = "LIGHT.BILLS"
$ TYPE 'LIT'
```

The TYPE command expects a file specification. The apostrophes indicate that LIT is a symbol that must be evaluated. If you had not used apostrophes, DCL would have looked for a file called LIT.LIS (LIS is the default file type for the TYPE command).

Use the apostrophe to request symbol substitution on the right side of a := (string assignment) statement. For example:

```
$ NAME := REPORT
$ FILE := 'NAME'.DAT
$ SHOW SYMBOL FILE
  FILE = "REPORT.DAT"
```

The value for NAME is substituted so that FILE becomes REPORT.DAT.

When you use apostrophes to request symbol substitution, you cannot continue the line (with the hyphen continuation character) in the middle of the value that is being substituted.

7.2.1.1 Concatenation of Symbol Names

You can concatenate two or more symbol names. Place apostrophes around each symbol name. For example:

```
$ NAME = "MYFILE"
$ TYPE = ".DAT"
$ PRINT 'NAME''TYPE'
```

The PRINT command prints a copy of MYFILE.DAT.

7.2.1.2 Substitution Within Character Strings

You can request symbol substitution within a quoted character string. Place two apostrophes before the symbol name and one apostrophe after it. For example:

```
$ MESSAGE = "Creating file ''NAME'.DAT"
```

If the current value of the symbol NAME is FRED, then MESSAGE has the following value:

```
Creating file FRED.DAT
```

7.2.2 The Ampersand (&)

The command interpreter also recognizes the ampersand as a substitution operator. In many cases, the apostrophe and the ampersand are functionally equivalent. For example:

```
$ TYPE 'NAME'
$ TYPE &NAME
```

In the first command, the command interpreter replaces the symbol NAME with its current value during the first phase of command processing (scanning). The second command replaces the symbol NAME with its current value during the second phase of command processing (parsing). The result is the same, even though the methods are different.

Ampersands are most effective as substitution operators when they are used with apostrophes to affect the order in which substitution is performed. For example:

```
$ P1 = "FRED.DAT"
$ COUNT = 1
$ TYPE &P'COUNT'
```

First, the command interpreter evaluates the symbol enclosed by apostrophes (COUNT). The result is as follows:

```
TYPE &P1
```

Next, the command interpreter evaluates the symbol preceded by an ampersand (P1). The result is as follows:

```
TYPE FRED.DAT
```

Suppose you had used apostrophes with both P and COUNT, as in the following example:

```
$ TYPE 'P''COUNT'
```

Working left-to-right, the command interpreter attempts to evaluate P. P is not a defined symbol, so DCL gives it a null value. Next, it evaluates the symbol COUNT. The result is as follows:

```
PRINT 1
```

The action the command interpreter takes when a symbol is undefined depends on the context of the command. For more information, see Section 7.5.

In the next example, A is equated to the current value of B:

```
$ B = "MYFILE.DAT"
$ A = "&B"
$ TYPE 'A'
```

The ampersand does not cause symbol substitution when it is used inside quotation marks. Therefore, when the assignment is made, the value of B is not substituted. However, the TYPE command displays MYFILE.DAT. This occurs because the command interpreter first substitutes the value &B for A. Next, it substitutes MYFILE.DAT for the symbol &B. If you were to redefine B, the result of the TYPE command would change accordingly.

Observe the following rules:

- Place the ampersand before, but not after, the symbol name.
- An ampersand must follow a delimiter (any blank or special character).
- You cannot use ampersands to request substitution within character strings enclosed in quotation marks.
- You cannot use ampersands to concatenate two or more symbol names.

7.3 The Three Phases of Command Processing

The command interpreter performs symbol substitution in three phases, as follows:

1 **Command input scanning** (also called the lexical input phase)

From left to right, the command interpreter evaluates symbols preceded by apostrophes. Symbols that are preceded by single apostrophes are translated iteratively, as described in Section 7.4.1. Symbols preceded by double apostrophes are not translated iteratively.

2 **Command parsing**

- The command interpreter analyzes the command line. It checks the first item on the line to see if it is a symbol. If it is, it is evaluated.
- From left to right, the command interpreter evaluates symbols preceded by ampersands.

Symbol substitution during this phase is not iterative.

3 **Expression evaluation**

- The command interpreter evaluates symbols that are preceded by the DEPOSIT, EXAMINE, IF, and WRITE commands.
- The command interpreter evaluates symbols within lexical functions.

Symbol substitution during this phase is not iterative.

Note: The command interpreter does not scan any lines that are read as input data by commands or programs executed within a command procedure. Therefore, the command interpreter does not perform symbol substitution within these data lines. For example:

```
$ RUN AVERAGE
55
57
9999
```

The program AVERAGE reads 55, 57, and 9999 from SYS$INPUT (the command input stream). These data lines are never read by the command interpreter. If you enter symbol names as input, they are not evaluated.

7.4 Repetitive and Iterative Substitution

Symbol substitution can be repetitive or iterative:

- Repetitive substitution results when more than one type of substitution occurs in a single command line.
- Iterative substitution occurs when the command interpreter examines a substituted value to see if the value itself is a symbol. Iterative substitution occurs only when symbols preceded by apostrophes are translated during the first phase of command processing.

The following sections describe iterative substitution.

7.4.1 First Phase

When you use an apostrophe to request symbol substitution, the command interpreter performs iterative substitution during the first phase of command processing.

Substitution using apostrophes is not iterative, however, when a symbol is included in a quoted character string. For example:

```
$ MAC = "5"
$ A = "'MAC'"
$ B = 'A'
$ SHOW SYMBOL B
  B = "5"
```

After the statement B = 'A' the resulting value of the symbol B is 5. The explanation follows:

1. The symbol name A is enclosed in apostrophes, so it is replaced with its current value ('MAC').
2. Because this value ('MAC') is also enclosed in apostrophes, the command interpreter replaces MAC with its current value (5).
3. Because this value (5) has no apostrophes, the first phase of command processing is complete. No further substitution is required during the second or third phases. Therefore, 5 is the final value given to the symbol name B.

Note, however, what happens when you include A in a quoted character string:

```
$ B = "''A'"
$ SHOW SYMBOL B
  B = "'MAC'"
```

In this case, B has the value 'MAC'. The symbol name A is replaced only once, because substitution is not iterative within quoted character strings.

7.4.2 Second Phase

The command interpreter performs iterative substitution automatically only when an apostrophe is in the command line. In some cases, you may want to nest command synonym definitions, as follows:

```
$ MAC = "TYPE A.B"
$ EXEC = "'MAC'"
$ EXEC
```

In this example, when EXEC is processed, the command interpreter performs substitution only once. The result is the string 'MAC'. The command interpreter displays an error message because it does not recognize MAC as a command.

This error occurs because, during the first phase of command processing, no substitution is performed (the string EXEC is not delimited by apostrophes). During the second phase, the string 'MAC' is substituted for EXEC because EXEC is the first value on the command line. This substitution is not iterative. Therefore, even though 'MAC' is delimited by apostrophes, no additional substitution is performed.

To use the command synonym EXEC correctly, enclose it in apostrophes, as shown below:

```
$ 'EXEC'
```

In this case, the symbol EXEC is evaluated during the first phase of command processing. Because this substitution is iterative, ('MAC') is also evaluated and the string TYPE A.B is substituted.

7.4.3 Third Phase

When the command interpreter analyzes an expression in a command, any symbols specified in the expression are replaced only once. You can, however, force iterative substitution by using an apostrophe or an ampersand in the expression. When you force iteration in this way, you must remember the following:

- The command interpreter performs all substitutions requested by apostrophes and ampersands before the command string is executed.
- Commands that automatically perform symbol substitution do so after the first and second phases of command processing.

The following example shows iterative substitution in an IF command.

```
$ P1 = "FRED.DAT"
$ COUNT = 1
$ IF P'COUNT' .EQS. "" THEN GOTO END
```

When the command interpreter scans this line, it replaces the symbol COUNT with its current value. The result is as follows:

```
IF P1 .EQS. "" THEN GOTO END
```

Because this string has no apostrophes, the command interpreter does not perform any more substitution. However, when the IF command executes, it automatically evaluates the symbol name P1 and replaces it with its current value.

Note, however, that if substitution does not result in a valid symbol name, the command fails. For example:

```
$ FILENAME = "A.B"
$ IF 'FILENAME' .NES. "" THEN TYPE 'FILENAME'
```

The command interpreter replaces the symbol FILENAME with its current value (A.B). The result is as follows:

```
IF A.B .NES.  "" THEN TYPE A.B
```

When the IF command executes the command line, A.B is not a valid symbol and an error occurs. For this IF command to be processed correctly, omit the apostrophes, as follows:

```
$ IF FILENAME .NES. "" THEN TYPE 'FILENAME'
```

7.5 Undefined Symbols

If a symbol is not defined when it is used in a command line, the command interpreter either displays an error message or replaces the symbol with a null string, depending on the context. The rules are as follows:

- During the first and second phases of command processing, the command interpreter replaces all undefined symbols that are preceded by apostrophes or ampersands with null strings.
- During the third phase of command processing, if the command interpreter finds an undefined symbol, it displays a warning message and does not finish processing.

The following example shows how the command interpreter processes an undefined symbol that is preceded by an apostrophe:

```
$ FILE := MYFILE'FILE_TYPE'
$ SHOW SYMBOL FILE
  FILE = "MYFILE"
$ PRINT 'FILE'
```

When the symbol FILE is created, the symbol FILE_TYPE is replaced with its current value. If FILE_TYPE is not defined, the command interpreter replaces FILE_TYPE with a null string. The absence of a file type in the file specification causes the PRINT command to use the default file type of LIS. Thus, the file specification is interpreted as MYFILE.LIS.

In the following example, the expression is evaluated during the third phase of command processing:

```
$ A = 1
$ C = A + B
%DCL-W-UNDSYM, undefined symbol - check validity and spelling
```

The symbol B is undefined so the command interpreter cannot evaluate the expression.

8 Protection

Devices, volumes, logical name tables, files, directories, mailboxes, common event flag clusters, global sections, and queues are some of the things you work with in the VMS operating system. In general, these are referred to as *system objects*. The operating system provides two mechanisms to control the access that users have to system objects:

- UIC-based protection—Every user has a *user identification code* (UIC) stored in the *user authorization file* (UAF). Each system object also has a UIC (the UIC of its owner) and a protection code that defines who is allowed what type of access. The relationship between your UIC and the object's UIC controls whether or not you have access to it.
- ACL-based protection—A system object can have an access control list (ACL) that specifies the access that a particular user or group of users are allowed. You can specify ACL-based protection for files, directories, devices, logical name tables, and global sections.

When determining whether or not you have access to a particular object, the system first checks to see if the object has an ACL. If there is not an ACL, or if the ACL does not explicitly allow or refuse access, the system uses UIC-based protection to determine access. (Even if the ACL denies access, the system may still grant access based on the SYSTEM and OWNER fields of the UIC-based protection. You can also use BYPASS, GRPPRV, READALL, or SYSPRV privilege to override ACL- and UIC-based protection.)

This chapter describes UIC-based protection. For more information on ACL-based protection, see the *Guide to VMS System Security*.

8.1 What is UIC-Based Protection?

Each user of the system has a UIC defined in the system UAF. Each system object also has a UIC (the UIC of its owner) and a protection code. The system compares your UIC to the UIC of a system object. This comparison reveals the relationship between you and the object. It tells what type of user you are in relationship to that object (for example, are you its owner). Next, the system checks the protection code of the object. The protection code specifies what type of access certain types of users have.

8.1.1 User Identification Code (UIC)

The system manager uses the AUTHORIZE utility to assign a UIC to each user. The UIC tells what group you belong to followed by your unique identification within that group. A UIC can be in either numeric or alphanumeric format.

The **numeric format** of a UIC is as follows:

```
[group,member]
```

where:

- *group* is the number of the group you belong to. It is an octal number in the range of 0 through 37776.
- *member* is your unique member number, an octal number in the range of 0 through 177776.

You can omit leading zeros when you specify group and member numbers in numeric format. The brackets are required.

The **alphanumeric format** of a UIC is as follows:

[member]

[group,member]

where:

- *group* is the name of the group you belong to. The group name is optional.
- *member* is your unique name within the group.

Group and member names in alphanumeric format can contain 1 to 31 alphanumeric characters and must contain at least one alphabetic character. The names can also include the dollar sign ($) and underscore (_). The brackets are required.

8.1.2 UIC Translation and Storage

Regardless of the format of the UIC, the system translates it to a 32-bit value that represents a group number and a member number; the high-order 16 bits contain the group number and the low-order 16 bits contain the member number. When translating an alphanumeric UIC, VMS equates the member part of the alphanumeric UIC to both the group and member parts of a numeric UIC. The resulting 32-bit numeric UIC is kept in the *system rights database* (a file that contains information about access rights).

This method of storing alphanumeric UICs dictates that member names must be unique and that no member can participate in more than one group. That is, each member name must be unique for each user on the system. For example, you could not have the two UICs [GROUP1,JONES] and [GROUP2,JONES] on the same system. The member JONES can have only one numeric UIC.

Every UIC has a group name associated with it. When the system translates an alphanumeric UIC that includes both a group and a member name, the system obtains the longword integer associated with the member. It then checks the group name against the member.

Because an alphanumeric UIC is equated to a numeric UIC in the system rights database, you can generally specify either format to refer to a user. For example, the following UIC specifications could all be valid for the user JONES:

```
[360,031]
[JONES]
[GROUP1,JONES]
```

Note: You can use either numeric or alphanumeric format in a DCL command that requires a UIC specification.

8.1.3 How the System Determines Access

When you attempt to access a system object, your UIC is compared to the owner UIC of the object. Once the two UICs are compared, you are put into one or more of the following *user categories*:

SYSTEM

- All users who have the system privilege (SYSPRV).
- Users with low group numbers, usually from 1 through 10 (octal). However, the exact range of system group numbers is determined by the system manager (with the SYSGEN parameter MAXSYSGROUP) when the system is generated, and may range as high as 37776 (octal). These group numbers are generally for system managers, security managers, system programmers, and operators.
- Users with the user privilege GRPPRV whose UIC group matches the group of the object's owner.
- For files on disk volumes, users whose UIC matches the owner UIC of the volume on which the file is located.

OWNER — The user with the same UIC as the user who created (and therefore owns) the object.

GROUP — All users, including the owner, who have the same group number in their UICs as the object's owner.

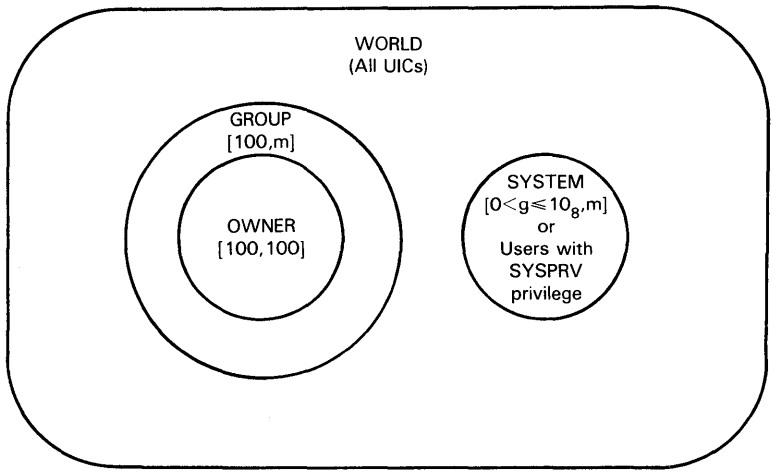
WORLD — All users, including those in the first three categories.

Figure 8–1 illustrates the relationships of these categories to each other.

Protection

8.1 What is UIC-Based Protection?

Figure 8–1 Illustrating User Categories with a UIC of [100,100]



g = Group Number
m = Member Number

NOTE: THE SYSTEM MANAGER CAN EXTEND THE SYSTEM GROUP NUMBER LIMIT TO 37776$_8$

ZK-778-82

The protection code determines what type of access each user category has. With UIC-based protection you can specify any of the following types of access:

- READ
- WRITE
- EXECUTE
- DELETE

CONTROL access is a fifth type of access that grants the user all the privileges of the object's actual owner. For example, if you have CONTROL access, you can change the protection and file characteristics, just as the owner could. CONTROL access is automatically granted to users in the system or owner categories. Users in the group or world categories never receive control access. Although you cannot specify CONTROL access in the standard UIC-based protection code, you can include it in an ACL.

The actual abilities conveyed by READ, WRITE, EXECUTE, DELETE, and CONTROL vary depending on the situation where they apply. For example, EXECUTE access permits very different operations depending on whether it is granted for file, directory, or volume access.

8.1.4 The Protection Code

The protection code lists each user category followed by the type of access that it has. For example:

```
$ SET PROTECTION=(SYSTEM:RWED,OWNER:RWED,GROUP:RE,WORLD:RE) SURVEY.DIR
```

This protection code specifies that anyone in the SYSTEM and OWNER categories has READ, WRITE, EXECUTE, and DELETE access to the file SURVEY.DIR. Anyone in the GROUP and WORLD categories has READ and EXECUTE access.

The following syntax rules apply to protection codes:

- When you specify a protection code, abbreviate access types to one character (R, W, E, or D). User categories can be entered in full or truncated to any number of characters. Separate each user category from its access types with a colon or an equal sign. If you specify more than one user category, separate the categories with commas and enclose the entire code in parentheses.
- You can specify the user categories and access types in any order. If you omit an access type for a user category, that category of user is denied that type of access. If you want to deny all access to a user category, specify the user category but omit the colon and do not list any access types. The following example denies all access to those in the WORLD category:

  ```
  $ SET PROTECTION=(SYSTEM:RWED,OWNER:RWED,GROUP:R,WORLD) HIRE.DAT
  ```

- When you omit a user category from a protection code applied to one or more files, or from a code specified for the default protection, the current access allowed for that category remains unchanged.
- When you omit a user category from a protection code applied to an entire volume, that category is denied all types of access. However, SYSTEM and OWNER always have access on magnetic tape volumes, regardless of the protection specified.

8.1.5 How the System Interprets a Protection Code

To determine access to an object, the system uses the object's protection code for each user category. The system checks user categories from outermost to innermost, in the following sequence:

1. WORLD
2. GROUP
3. OWNER
4. SYSTEM

You can access an object as soon as the system finds a user category that you fit into that gives you the access you have requested.

To deny access to a user category, be sure to deny access to all outer categories. For example, the following protection code denies DELETE access to a file's owner:

```
SYSTEM:RWED, OWNER:RW, GROUP:RW, WORLD:RWED
```

However, the owner can still delete the file because the owner is also a member of the WORLD category, which has DELETE access.

8.1.6 How Privileges Affect Protection

There are four system privileges that affect the access a user actually receives. They are as follows:

SYSPRV	A user with SYSPRV receives the access accorded to users in the SYSTEM category.
GRPPRV	A user with GRPPRV, whose UIC group matches the group of the owner of the object, receives the same access accorded to users in the SYSTEM category. Thus, the user with GRPPRV is able to manage a group's files.
BYPASS	A user with BYPASS receives all types of access to the object, regardless of its protection.
READALL	A user with READALL receives READ and CONTROL access to the object, even if that access is denied by the ACL- or UIC-based protection. In addition, the user may receive any other access that is granted through the protection code.

If a user holds any of these privileges, the outcome of the protection check may be quite different from your expectations. For example, a user with BYPASS privilege can delete any file on the system, despite the protection any file might have.

8.2 Establishing and Changing UIC-Based Protection

This section describes how to establish or change the UIC-based protection for devices, queues, volumes, directories, logical name tables, global sections, and files.

8.2.1 Devices

UIC-based protection on record-oriented devices must be reestablished every time the system is booted. Set device protection with the following command:

SET PROTECTION=(code)/DEVICE device-name[:]

When applied to devices, access types have the following meanings:

READ	The right to issue read requests to the device.
WRITE	The right to issue write requests to the device.
CONTROL	The right to change the device ACL.

8.2.2 Queues

UIC-based protection lets you restrict the types of jobs and users for a particular queue. Set queue protection with any of the following commands:

```
INITIALIZE/QUEUE/PROTECTION=(code)
START/QUEUE/PROTECTION=(code)
SET QUEUE/PROTECTION=(code)
```

When applied to queues, access types have the following meanings:

READ	The right to display the attributes of a job.
WRITE	The right to submit jobs to the queue.
EXECUTE	The right to act as operator for the queue, the ability to affect any jobs in the queue.
DELETE	The right to delete a job.

8.2.3 Volumes

Volume protection is coded into the home block of the disk or tape when it is mounted. It can be specified or defaulted from the device protection code. Set volume protection with any of the following commands:

```
INITIALIZE
MOUNT
SET VOLUME
```

For disk volumes, the system provides protection at the file, directory, and volume levels. For tape volumes, the system provides protection only at the volume level. The protection applied to a magnetic tape volume applies equally to all files on the volume.

When applied to volumes, access types have the following meanings:

READ	The right to examine, print, or copy files on a volume.
WRITE	The right to modify or to write existing files on a volume.
EXECUTE	The right to create files on the volume and write into them.
DELETE	The right to delete files on the volume.
CONTROL	The right to change the protection and ownership of the volume.

If you do not specify the protection when a volume is initialized, all users have READ and WRITE access. Moreover, system users and the owner are always given both READ and WRITE access, regardless of what you specify in a protection code. Granting a user category WRITE access automatically permits READ access.

Keep the following points in mind when setting the protection for a magnetic tape volume:

- EXECUTE and DELETE access are not valid for magnetic tapes.
- For magnetic tapes mounted with the /FOREIGN qualifier, system users and the owner are always given logical and physical I/O access in addition to READ and WRITE access, regardless of what you specify in the protection code.

- File protection on a given magnetic tape can be changed only if the tape is reinitialized.

8.2.4 Directories

Each directory file has a protection associated with it. Directory protection can be specified or defaulted from the directory level above it. Set directory protection with either of the following commands:

```
CREATE/DIRECTORY/PROTECTION=(code)
SET PROTECTION=(code)
```

When applied to directories, access types have the following meanings:

READ	The right to examine or list the directory file.
WRITE	The right to modify or write to the directory file
EXECUTE	The right to look up files in the directory if you specify the file name
DELETE	The right to delete the directory file

READ access lets you display the contents of the directory file with the DIRECTORY command. You can use wildcards (explicitly or implicitly). In addition, you can access any file cataloged in the directory unless the file's protection denies you access. However, if a directory denies you READ access, you cannot look up or access even those files that permit access to users in your group. (It is possible to access files without using the directory in which they are listed through suitable programming techniques. To guarantee protection, therefore, individual files should also be protected.)

WRITE access lets you write to the directory file. You must have both READ and WRITE access to a directory to create files in it, rename files, or perform any file operation that involves changes to the directory file.

EXECUTE access has a special meaning when it is applied to directories. EXECUTE access lets you use the DIRECTORY command to look up files that you can identify by name. In addition, if you do not perform an operation that modifies the directory file, you can access files that are not protected against users in your category. However, you cannot list all the entries in the directory by using wildcards. Therefore, EXECUTE access provides some, but not all of the operations that READ provides.

DELETE access lets you delete a directory file. Before you delete a directory file, do the following:

1. Remove all the files that are in it.
2. Use the SET PROTECTION command to assign DELETE access to the OWNER category of the directory file.

Directory protection can override the protection of individual files in the directory. Make sure your files are adequately protected at both the directory and file level.

8.2.5 Files

Each file on a disk has its own protection code. You can specify a protection code when you create a file or change the protection for an existing file. Set file protection with either of the following commands:

```
COPY/PROTECTION=(code)
SET PROTECTION=(code)
```

If you do not define a protection code for a file when you create it, the system applies a default protection. If a version of the file already exists, protection is taken from the previous version of the file. For a new file, the protection is determined in one of two ways:

- If the directory where the file is to be cataloged has an associated access control list that specifies the DEFAULT_PROTECTION entry, then the specified protection is used.
- If the directory does not have an associated access control list, then the default process protection is used. (The default process protection is established with the SET PROTECTION/DEFAULT command, or by default when you log in.)

Use the following DCL commands to change or display file protection:

SHOW PROTECTION	Displays the current default protection.
SET PROTECTION/DEFAULT	Changes the default protection applied to files that you create during a terminal session.
DIRECTORY/PROTECTION	Displays the current protection associated with a specific file or group of files.

When applied to files, access types have the following meanings:

READ	The right to examine, print, or copy the file.
WRITE	The right to modify or write to the file.
EXECUTE	The right to execute a file that contains an executable program image or DCL command procedure.
DELETE	The right to delete the file.
CONTROL	The right to change the protection and file characteristics of the file.

Note the following:

- READ access also implies EXECUTE access.
- To open a file for a write operation, you must have both READ and WRITE access. This is because the VMS operating system does not support write-only files.
- To delete a file you must have DELETE access to both the file and the directory that contains the file.

8.2.6 Global Sections

UIC-based protection on global sections, except those backed by disk files, must be reestablished every time the system is booted. If the global section is backed by a disk file, the section protection is derived from the disk file. Changing the file protection changes the section protection.

For PFN and page file global sections, you set the protection in the $CRMPSC system service call that creates the section. You cannot change the protection after the section is created.

When applied to global sections, access types have the following meanings:

READ	The right to map the section for read access.
WRITE	The right to map the section for write access.
EXECUTE	The right to map the section for execute access (available only to privileged software).
CONTROL	The right to change the access control list (applies only to PFN and page file global sections).

8.2.7 Logical Name Tables

UIC-based protection on logical name tables must be reestablished every time the system is booted. Set logical name table protection by applying the protection argument to the $CRELNT system service call or with the following command:

```
CREATE/NAME_TABLE/PROTECTION=(code) table-name
```

You cannot change the protection on an existing logical name table.

When applied to logical name tables, access types have the following meanings:

READ	The right to look up logical names in the table.
WRITE	The right to create and delete logical names in the table.
DELETE	The right to delete the table.
CONTROL	The right to change the logical name table ACL.
ENABLE	The right to create a shareable logical name table.

A VMS Process Privileges and Resource Quotas

Tables A–1 and A–2 summarize the full set of process privileges and resource quotas. The system manager sets each user's privileges and resource quotas in the user authorization file (UAF). If you need special quotas or privileges to use DCL commands, the command descriptions in the *VMS DCL Dictionary* note these restrictions.

Table A–1 Process Privileges

Privilege	Operations Permitted
ACNT	Create a process or subprocess with accounting disabled (RUN command and SYS$CREPRC system service)
ALLSPOOL	Allocate a spooled device (ALLOCATE command and SYS$ALLOC system service)
ALTPRI	Increase base priority and create processes with higher priorities (SYS$SETPRI and SYS$CREPRC system services)
BUGCHK	Make BUGCHK error log entries
BYPASS	Access all objects bypassing protection
CMEXEC	Change mode to executive (SYS$CMEXEC system service)
CMKRNL	Change mode to kernel (SYS$CMKRNL system service)
DETACH	Create a detached process (SYS$CREPRC system service) with arbitrary UIC
DIAGNOSE	Run online diagnostic programs and read messages written to the error log file
EXQUOTA	Exceed disk quotas
GROUP	Affect other processes in the same group (SET QUEUE, DELETE/ENTRY, STOP/ENTRY, and SET PROCESS commands; SYS$SUSPND, SYS$RESUME, SYS$DELPRC, SYS$SETPRI, SYS$WAKE, SYS$SCHDWK, SYS$CANWAK, SYS$FORCEX, and SYS$GETJPI system services)
GRPNAM	Create and delete group logical names (DEFINE, DEASSIGN, and MOUNT commands; SYS$CRELNM and SYS$DELLNM system services)
GRPPRV	Access protected files and other objects within the same group as a system user, and change the protection on files and other objects within the same group
LOG_IO	Perform logical I/O operations (SYS$QIO system service)
MOUNT	Perform the mount volume I/O function (SYS$QIO system service)
NETMBX	Perform DECnet operations
OPER	Set devices spooled, control queues, control public volumes, broadcast messages, and perform other system-wide operations
PFNMAP	Map to physical memory and I/O registers

Table A–1 (Cont.) Process Privileges

Privilege	Operations Permitted
PHY_IO	Perform physical I/O operations (SYS$QIO system service)
PRMCEB	Create and delete permanent common event flag clusters (SYS$ASCEFC and SYS$DLCEFC system services)
PRMGBL	Create global sections (SYS$CRMPSC system service) and install global sections (also requires CMKRNL and SYSGBL privileges)
PRMMBX	Create and delete permanent mailboxes (SYS$CREMBX and SYS$DELMBX system services)
PSWAPM	Disable and enable swapping (RUN command; SYS$CREPRC and SYS$SETSWM system services)
READALL	Allow read and control access to all objects
SECURITY	Perform security-related activities such as enabling or disabling security audits and setting the system password
SETPRV	Grants a process any privilege
SHARE	Assign a channel to a device even if the channel is allocated to another device
SHMEM	Create global sections and mailboxes in multiport memory (also requires the appropriate PRMGBL, PRMMBX, SYSGBL, and TMPMBX privileges)
SYSGBL	Create system global sections (SYS$CRMPSC) and install known images (also requires CMKRNL and PRMGBL privileges)
SYSLCK	Lock system-wide resources (SYS$ENQ system service)
SYSNAM	Create and delete system logical names (DEFINE, DEASSIGN, and MOUNT commands; SYS$CRELNM and SYS$DELLNM system services)
SYSPRV	Access protected files and other objects as a system user and change the protection on files and other objects
TMPMBX	Create temporary mailboxes (SYS$CREMBX system service)
VOLPRO	Initialize a volume with a different UIC, override an expiration date, mount a volume foreign, and override volume protection (affecting system volumes also requires SYSNAM privilege)
WORLD	Affect all other processes (SET QUEUE, DELETE/ENTRY, STOP/ENTRY, and SET PROCESS commands; SYS$SUSPND, SYS$RESUME, SYS$DELPRC, SYS$SETPRI, SYS$WAKE, SYS$SCHDWK, SYS$CANWAK, SYS$FORCEX, and SYS$GETJPI system services)

Table A–2 Resource Quotas

Name	Quota
ASTLM	AST (asynchronous system trap) limit
BIOLM	Buffered I/O limit
BYTLM	Buffered I/O byte count (buffer space) quota
CPUTIME	CPU time limit
DIOLM	Direct I/O limit
ENQLM	Enqueue limit
FILLM	Open file quota
JTQUOTA	Initial byte quota for job logical name table
MAXACCTJOBS	Maximum active processes for an account
MAXDETACH	Maximum detached processes for a user name
MAXJOBS	Maximum active processes for a user name
PGFLQUOTA	Paging file quota
PRCLM	Subprocess quota
SHRFILLM	Maximum number of open shared files
TQELM	Timer queue entry quota
WSDEFAULT	Default working set size
WSEXTENT	Working set extent quota
WSQUOTA	Working set size quota

B DEC Multinational Character Set

Figure B–1 DEC Multinational Character Set and Hexadecimal Values

HEX Code	ASCII Char.	HEX Code	ASCII Char.	HEX Code	ASCII Char.	HEX Code	ASCII Char.
00	NUL	20	SP	40	@	60	`
01	SOH	21	!	41	A	61	a
02	STX	22	"	42	B	62	b
03	ETX	23	#	43	C	63	c
04	EOT	24	$	44	D	64	d
05	ENQ	25	%	45	E	65	e
06	ACK	26	&	46	F	66	f
07	BEL	27	'	47	G	67	g
08	BS	28	(	48	H	68	h
09	HT	29	)	49	I	69	i
0A	LF	2A	*	4A	J	6A	j
0B	VT	2B	+	4B	K	6B	k
0C	FF	2C	,	4C	L	6C	l
0D	CR	2D	-	4D	M	6D	m
0E	SO	2E	.	4E	N	6E	n
0F	SI	2F	/	4F	O	6F	o
10	DLE	30	0	50	P	70	p
11	DC1	31	1	51	Q	71	q
12	DC2	32	2	52	R	72	r
13	DC3	33	3	53	S	73	s
14	DC4	34	4	54	T	74	t
15	NAK	35	5	55	U	75	u
16	SYN	36	6	56	V	76	v
17	ETB	37	7	57	W	77	w
18	CAN	38	8	58	X	78	x
19	EM	39	9	59	Y	79	y
1A	SUB	3A	:	5A	Z	7A	z
1B	ESC	3B	;	5B	[	7B	{
1C	FS	3C	<	5C	\	7C	|
1D	GS	3D	=	5D	]	7D	}
1E	RS	3E	>	5E	^	7E	~
1F	US	3F	?	5F	_	7F	DEL

ZK-820-82

C DCL Character Set

Table C–1 DCL Character Set

Symbol	Name	Meaning
@	At sign	Places the contents of a command procedure file in the command input stream.
:	Colon	Device name delimiter in a file specification. A double colon (::) is a node name delimiter. A colon also acts as a qualifier delimiter. It separates a qualifier name from its value.
/	Slash	Qualifier prefix.
+	Plus sign	Parameter separator. With some commands it acts as a parameter concatenator. The plus sign is also recognized as a string concatenation operator, a unary plus sign, and an addition operator in a numeric expression.
,	Comma	List element separator for parameters or argument lists.
-	Hyphen	Continuation character. The hyphen is also recognized as a string reduction operator, a unary minus sign, a subtraction operator in a numeric expression, and a directory-searching wildcard character.
( )	Parentheses	List delimiters for argument list. Parentheses are also used to indicate the order of operations in a numeric expression.
[ ]	Square brackets	Directory name delimiters in a file specification. Equivalent to angle brackets.
< >	Angle brackets	Directory name delimiters in a file specification. Equivalent to square brackets.
?	Question mark	Help character.
&	Ampersand	Execution-time substitution operator. Otherwise, a reserved special character.
\	Backslash	Reserved special character.
=	Equal sign	Qualifier value delimiter. It separates a qualifier name from its argument.
^	Circumflex	Reserved special character.
#	Pound sign	Reserved special character.
*	Asterisk	Wildcard character in a file specification. The asterisk is also used as a multiplication operator in a numeric expression and as an abbreviation delimiter in a symbol definition.
'	Apostrophe	Substitution operator.

Table C–1 (Cont.) DCL Character Set

Symbol	Name	Meaning
.	Period	File type and version number delimiter in a file specification. Also used as a subdirectory delimiter.
;	Semicolon	Version number delimiter in a file specification.
%	Percent sign	Wildcard character in a file specification. Also used as a radix operator.
!	Exclamation point	Indicates a comment.
"	Quotation mark	Literal string delimiter.

Index

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Reader's Comments

VMS DCL Concepts Manual
AA-LA10A-TE

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

I would like to see more/less _____

_____

_____

What I like best about this manual is _____

_____

_____

What I like least about this manual is _____

_____

_____

I found the following errors in this manual:

Page	Description
_____	_____
_____	_____
_____	_____
_____	_____
_____	_____

Additional comments or suggestions to improve this manual:

_____

_____

_____

_____

I am using **Version** _____ of the software this manual describes.

Name/Title _____ Dept. _____

Company _____ Date _____

Mailing Address _____

_____ Phone _____

-- Do Not Tear - Fold Here and Tape ----------

digital™

No Postage
Necessary
if Mailed
in the
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01–3/J35 110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

-- Do Not Tear - Fold Here ----------