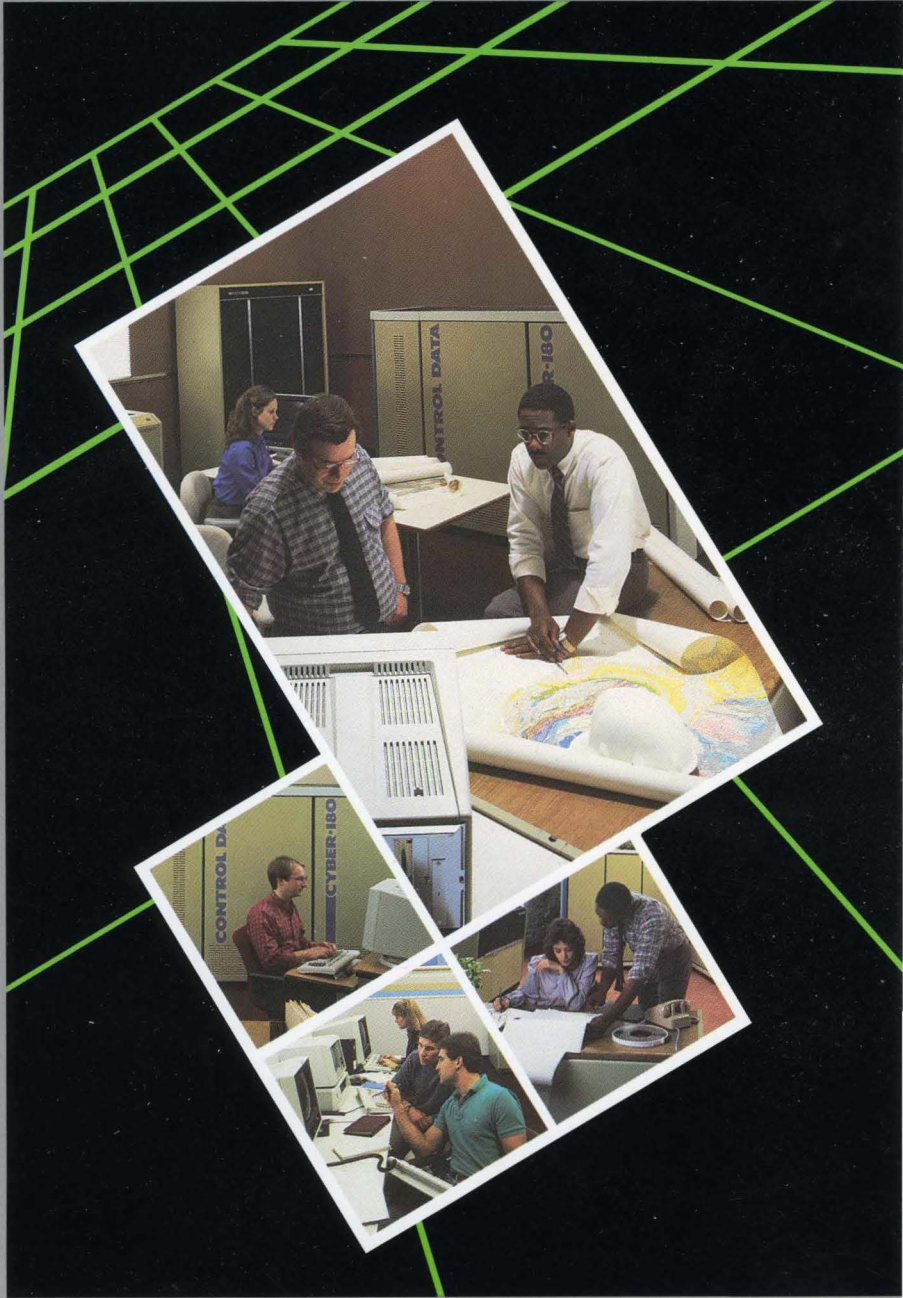


NOS/VE System Usage



NOS/VE System Usage

This product is intended for use only as described in this document. Control Data cannot be responsible for the proper functioning of undescribed features and parameters.

Manual History

Revision	System Version	PSR Level	Date
A	1.0.2		February 1984
B	1.1.1	613	August 1984
C	1.1.3	644	October 1985
D	1.2.1	664	September 1986
E	1.2.1	670	December 1986
F	1.2.2	678	April 1987
G	1.2.3	688	September 1987
H	1.3.1	700	April 1988

This manual merges the topics previously described in both the SCL for NOS/VE Language Definition and SCL for NOS/VE System Interface manuals. Both manuals are replaced with this manual, the NOS/VE System Usage manual.

In addition, the command and format descriptions formerly in SCL for NOS/VE Language Definition and SCL for NOS/VE System Interface now reside only in the NOS/VE Commands and Functions manual, formerly titled SCL for NOS/VE Quick Reference.

This manual also documents the following feature-related changes for NOS/VE version 1.3.1 at PSR 700:

- The `ADMINISTER_USER` utility has been replaced with the `ADMINISTER_VALIDATION` utility. `ADMINISTER_VALIDATION` is first discussed in chapter 3, Interactive Sessions.
- `SET_PASSWORD` has been changed to `CHANGE_LOGIN_PASSWORD` and two new parameters, `EXPIRATION_INTERVAL` and `EXPIRATION_DATE` have been added. This is discussed in chapter 3.

©1984, 1985, 1986, 1987, 1988 by Control Data Corporation
All rights reserved.
Printed in the United States of America.

- `DELETE_CATALOG` has been enhanced to allow a user to delete a non-empty catalog. `DELETE_CATALOG` is discussed in chapter 4, Catalog and File Management.
- Users wishing to attach a file for `READ` and/or `EXECUTE` access may now do so without sharing its file position with other, concurrent attachments of the file. This capability is called Private Read and is discussed in chapter 4.
- Job classes have been enhanced to allow users to select a particular job class and to allow sites to restrict users from using given job classes based on several different criteria. This is discussed in chapter 6, Job Management.
- `$JOB_DEFAULT` is a new function that returns information on a the default values a site establishes for a user's job attributes. The description of default job attributes is found in chapter 6. The `$JOB_DEFAULT` function is described in the NOS/VE Commands and Functions manual.
- ANSI standard X3.27 levels 3 and 4 have been added to NOS/VE's support of ANSI labelled tapes. This support is discussed in chapter 11, Tape Management.
- `INCLUDE_SMALL_CYCLES` has been added to the `BACKUP_PERMANENT_FILE` utility. This utility is discussed in chapter 12, Backup and Restore Utilities.

Appendixes A and B have been updated as needed.

This edition obsoletes all previous editions.

Contents

About This Manual	11	Entering Commands Interactively	3-9
The NOS/VE User Manual Set	12	Terminal Disconnects	3-19
Conventions	14	Reestablishing Terminal Connections	3-19
Submitting Comments	15	Job Time Limits	3-20
CYBER Software Support Hotline	15	Displaying Your User Validation	3-21
NOS/VE Concepts	1-1	Changing Your User Validations	3-23
NOS/VE Jobs	1-1	Getting Help On Line	3-24
The System Command Language	1-1	Using Online Manuals	3-28
The File System	1-2	Catalog and File Management	4-1
Working Environments	1-3	NOS/VE File and Catalog Structures	4-1
NOS/VE Attributes	1-3	Referencing File Names	4-5
Networks	1-4	Creating Catalogs and Files	4-13
Online Help	1-4	Catalog and File Permits	4-21
Dual-State Systems	1-4	Attaching and Detaching Permanent Files	4-30
The System Command Language	2-1	Using Files with Passwords	4-39
Input to SCL	2-2	NOS/VE File Attributes	4-40
Using NOS/VE Commands, Utilities, and Functions	2-11	Copying Files	4-64
Data Representation	2-29	Comparing Files	4-80
Assignment Statements	2-32	Displaying the Representation of Data in a File	4-82
Expressions	2-33	Managing Remote Files and Catalogs	4-84
Control Statements	2-34	Command and SCL Procedure Execution	5-1
Examples of Statements	2-35	Command Lists	5-2
Procedures	2-36	Command List Entries	5-3
Interactive Sessions	3-1		
Networks	3-2		
Logging In and Out	3-4		

Command List Search		Controlling the Flow of	
Modes	5-7	a Command Stream . .	8-10
Changing the Command		Executing Statement	
List	5-8	Lists Conditionally	
Displaying the Command		(IF/IFEND)	8-15
List	5-12	Condition Processing . .	8-16
Displaying Command		Suspending Command	
List Entries.	5-13	Processing (WAIT) . . .	8-26
Job Management	6-1	Changing System	
Overview of NOS/VE		Environments.	8-26
Jobs	6-1	Writing SCL Procedures	
Job Attributes	6-12	and Command Utilities. .	9-1
Submitting Batch Jobs .	6-25	Commands and	
Terminating Jobs	6-32	Functions Used in	
Displaying Job Status .	6-32	Procedure Writing	9-3
Managing Tasks	6-33	Creating Procedures . . .	9-18
Managing Job Output .	6-38	Procedure Calling	
Using Standard and Job		Environments.	9-20
Files.	6-57	Procedure Format	9-21
Managing Job Logs . . .	6-64	Defining Procedure	
Displaying Job Resource		Parameters	9-24
Limits.	6-70	Sample Procedure	9-42
Setting Multiprocessing		Creating SCL Command	
Options	6-71	Utilities.	9-45
Setting Job Sense		Writing Command	
Switches.	6-72	Utilities.	9-46
SCL Variables, Types,		Permitting Others to	
and Expressions.	7-1	Use Your Utilities. . . .	9-60
Creating and Using		Using Your Utility	9-62
Variables	7-2	Formatting SCL	
SCL Types	7-22	Procedures	9-67
Expressions	7-37	Dual-State File Access .	10-1
SCL Command Streams		File Access	
and Condition		Requirements.	10-1
Processing	8-1	Getting a File from a	
Block Structure	8-2	Dual-State Partner	
Structuring a Command		System	10-2
Stream	8-3	Replacing a File on a	
		Dual-State Partner	
		System	10-2

Setting Link Attributes Using ADMINISTER_ VALIDATION.	10-3	Glossary	A-1
Preserving NOS/VE File Attributes.	10-4	Related Manuals	B-1
Tape Management	11-1	Ordering Printed Manuals.	B-1
Reserving and Releasing Tape Units	11-1	Accessing Online Manuals.	B-1
Requesting Magnetic Tapes	11-4	Character Set	C-1
Rewinding Tape Files .	11-5	ASCII Character Set . . .	C-1
NOS/VE Labelled Tape Support	11-5	EBCDIC Character Set .	C-5
Using Labelled Tapes .	11-16	ANSI Tape Label Formats	D-1
Using Unlabelled Tape Files.	11-32	Required Labels	D-4
Backup and Restore Utilities	12-1	Optional Labels	D-25
Overview	12-1	User Labels	D-26
Backup Operations . . .	12-5	Format Effectors	E-1
Restore Operations . . .	12-19	Vertical Spacing Characters	E-1
Terminal Management .	13-1	Vertical Form Unit Loading	E-3
Attribute Overview . . .	13-1	SCL Language Syntax . . .	F-1
Network Dependencies .	13-3	Metalanguage Symbols . .	F-1
Managing Terminal Attributes.	13-4	SCL Language Syntax . .	F-3
Managing Connection Attributes.	13-23	Calls to Commands	F-5
Terminal Input	13-41	SCL Procedure Syntax . .	G-1
Terminal Output	13-46	Basic Syntax	G-1
Micro File Transfers . .	14-1	Expressions	G-4
Using CONNECT	14-1	Old Commands	H-1
Using XMODEM	14-6	Index	Index-1
KERMIT-VE	14-18		
Desktop/VE	14-21		

Figures

4-1. Master Catalog Structure	4-4	9-1. Sample SCL Procedure	9-43
4-2. Two Family File Hierarchy	4-9	9-2. Sample SCL Command Utility	9-50
4-3. Block and Record Types.	4-70	9-3. Unformatted SCL Procedure	9-68
7-1. Job Block Structure .	7-15	9-4. Formatted SCL Procedure	9-69
8-1. CYCLE Statement Use.	8-11	E-1. Vertical Forms Unit Load Image	E-4
8-2. EXIT Statement Use .	8-13	F-1. Calls to Commands . .	F-5

Tables

2-1. Characters in SCL Names	2-7	11-1. Conditions for Writing HDR Labels . . .	11-14
3-1. Summary of User Controlled Interrupts . . .	3-13	11-2. Default Tape Label Attributes	11-24
3-2. Summary of Online Help Commands.	3-24	12-1. Summary of Backup Utility Subcommands. . .	12-17
3-3. Summary of Screen Mode Operations	3-28	12-2. Summary of Restore Utility Subcommands. . .	12-27
3-4. Summary of Line Mode Subcommands . . .	3-30	13-1. Terminal Attribute Network Applicability . .	13-18
4-1. Changing Attributes of Old Files.	4-55	13-2. Terminal Attribute Editing Modes.	13-21
6-1. Standard Files and Initial Connections	6-58	13-3. NOS/VE Initial Values at the Default Level.	13-34
7-1. Scope of Variables . .	7-14	13-4. Connection Attribute Network Applicability . .	13-36
7-2. Status Variable Fields	7-32	13-5. Connection Attribute Editing Modes.	13-39
7-3. Operand and Operator Combinations	7-45	14-1. Configuration File Entries.	14-16
9-1. Commands Used in Procedure Writing	9-3	B-1. Related Manuals	B-2
9-2. Topical List of NOS/VE Functions	9-5	C-1. ASCII Character Set .	C-2
9-3. Utilities and Terminators	9-72	C-2. ASCII to EBCDIC Conversion for Tapes. . . .	C-6

D-1. Label Characteristics	D-2	D-7. EOVI - First	
D-2. VOL1 - Volume		End-of-Volume Label	D-23
Header Label (1)	D-6	D-8. EOVI2 - Second	
D-3. HDR1 - First File		End-of-Volume Label	D-24
Header Label (1)	D-9	E-1. Vertical Spacing	
D-4. HDR2 - Second File		Characters	E-1
Header Label (1)	D-16	F-1. SCL Metalanguage	
D-5. EOF1 - First		Conventions	F-1
End-of-File Label	D-20	H-1. NOS/VE Old	
D-6. EOF2 - Second		Commands	H-1
End-of-File Label	D-21		

About This Manual

This manual describes the command interface to the CONTROL DATA® Network Operating System/Virtual Environment (NOS/VE) using the System Command Language (SCL). It provides discussions on system access, interactive processing, file and catalog management, job management, tape file management, and terminal attributes.

This manual also describes the complete SCL language specification including language elements, expressions, variables, command stream structuring, and procedure creation.

A complete command reference description of the commands and functions discussed in this manual can be found in the NOS/VE Commands and Functions manual. In addition, command and function parameter information can be obtained online using the DISPLAY_COMMAND_INFORMATION and DISPLAY_FUNCTION_INFORMATION commands.

This manual requires no previous knowledge of NOS/VE other than specific login information provided by your site. However, if you are a first time user of NOS/VE, or if you are new to a mainframe computer environment, you may want to read the Introduction to NOS/VE manual before reading this manual. The Introduction to NOS/VE is a tutorial designed to help you become familiar with some of the more general uses of NOS/VE.

If you are a new user of NOS/VE and you do not have access to the Introduction to NOS/VE manual, or if you simply wish to skip that manual and proceed with this manual, you should read this manual sequentially starting with chapter 1 and continuing on through the rest of the book. If you are already acquainted with NOS/VE, this manual can be used as a reference guide to the usage of NOS/VE.

The NOS/VE User Manual Set

This manual is part of a set of user manuals that describe the command interface to NOS/VE. The descriptions of these manuals follow:

Introduction to NOS/VE

Introduces NOS/VE and SCL to users who have no previous experience with them. It describes, in tutorial style, the basic concepts of NOS/VE: creating and using files and catalogs of files, executing and debugging programs, submitting jobs, and getting help online.

The manual describes the conventions followed by all NOS/VE commands and parameters, and lists many of the major commands, products, and utilities available on NOS/VE.

NOS/VE System Usage

Describes the command interface to NOS/VE using the SCL language. It describes the complete SCL language specification, including language elements, expressions, variables, command stream structuring, and procedure creation. It also describes system access, interactive processing, access to online documentation, file and catalog management, job management, tape management, and terminal attributes.

NOS/VE File Editor

Describes the EDIT_FILE utility used to edit NOS/VE files and decks. The manual has basic and advanced chapters describing common uses of the utility, including creating files, copying lines, moving text, editing more than one file at a time, and creating editor procedures. It also contains descriptions of subcommands, functions, and terminals.

NOS/VE Source Code Management

Describes the SOURCE_CODE_UTILITY, a development tool used to organize and maintain libraries of ASCII source code. Topics include deck editing and extraction, conditional text expansion, modification state constraints, and using the EDIT_FILE utility.

NOS/VE Object Code Management

Describes the CREATE_OBJECT_LIBRARY utility used to store and manipulate units of object code within NOS/VE. Program execution is described in detail. Topics include loading a program,

program attributes, object files and modules, message module capabilities, code sharing, segment types and binding, ring attributes, and performance options for loading and executing.

NOS/VE Advanced File Management

Describes three file management tools: Sort/Merge, File Management Utility (FMU), and keyed-file utilities. Sort/Merge sorts and merges records; FMU reformats record data; and the keyed-file utilities copy, display, and create keyed files (such as indexed-sequential files).

NOS/VE Terminal Definition

Describes the `DEFINE_TERMINAL` command and the statements that define terminals for use with full-screen applications (for example, the `EDIT_FILE` utility).

NOS/VE Commands and Functions

Lists the formats of the commands, functions, and statements described in the NOS/VE user manual set. A format description includes brief explanations of the parameters and an example using the command, function, or statement.

Conventions

The following conventions are used in this manual:

Boldface	In a format, boldface type represents names and required parameters.
<i>Italics</i>	In a format, italic type represents optional parameters.
UPPERCASE	In a format, uppercase letters represent reserved words defined by the system for specific purposes. You must use these words exactly as shown.
lowercase	In a format, lowercase letters represent values you choose.
Blue	In examples of interactive terminal sessions, blue represents user input.
Vertical bar	A vertical bar in the margin indicates a technical change.
Numbers	All numbers are decimal unless otherwise noted.

Submitting Comments

There is a comment sheet at the back of this manual. You can use it to give us your opinion of the manual's usability, to suggest specific improvements, and to report errors. Mail your comments to:

Control Data Corporation
Technology and Publications Division ARH219
4201 North Lexington Avenue
St. Paul, Minnesota 55126-6198

Please indicate whether you would like a response.

If you have access to SOLVER, the Control Data online facility for reporting problems, you can use it to submit comments about the manual. When entering your comments, use NVO (zero) as the product identifier. Include the name and publication number of the manual.

If you have questions about the packaging and/or distribution of a printed manual, write to:

Control Data Corporation
Literature and Distribution Services
308 North Dale Street
St. Paul, Minnesota 55103

or call (612) 292-2101. If you are a Control Data employee, call (612) 292-2100.

CYBER Software Support Hotline

Control Data's CYBER Software Support maintains a hotline to assist you if you have trouble using our products. If you need help not provided in the documentation, or find the product does not perform as described, call us at one of the following numbers. A support analyst will work with you.

From the USA and Canada: (800) 345-9903

From other countries: (612) 851-4131

NOS/VE Concepts 1

NOS/VE Jobs	1-1
The System Command Language	1-1
The File System	1-2
Working Environments	1-3
NOS/VE Attributes	1-3
Networks	1-4
Online Help	1-4
Dual-State Systems	1-4

NOS/VE is the Network Operating System/Virtual Environment for Control Data's CYBER 180 computer systems. This chapter presents an overview of some of the NOS/VE concepts presented in this manual. The chapters that follow in this manual discuss the concepts presented here in detail.

NOS/VE Jobs

When a program is executed on NOS/VE, a task is created. In order to begin creating tasks on the system, you must identify yourself to NOS/VE. This process of identification is called logging in. You tell NOS/VE you are done creating tasks by logging out.

A job is a series of tasks delimited by logging in and logging out.

There are two different kinds of jobs on NOS/VE: interactive and batch. When you log in to NOS/VE at a terminal, you are initiating an interactive job. When you are running an interactive job, you control the job by entering commands to the system on a line by line basis.

With batch jobs, you submit the entire job from login to logout to the system for execution. In this way, you need not spend large amounts of time at your terminal controlling each individual phase of the job. At the same time, of course, you do not have the immediacy of control with batch jobs that you have with interactive jobs.

The System Command Language

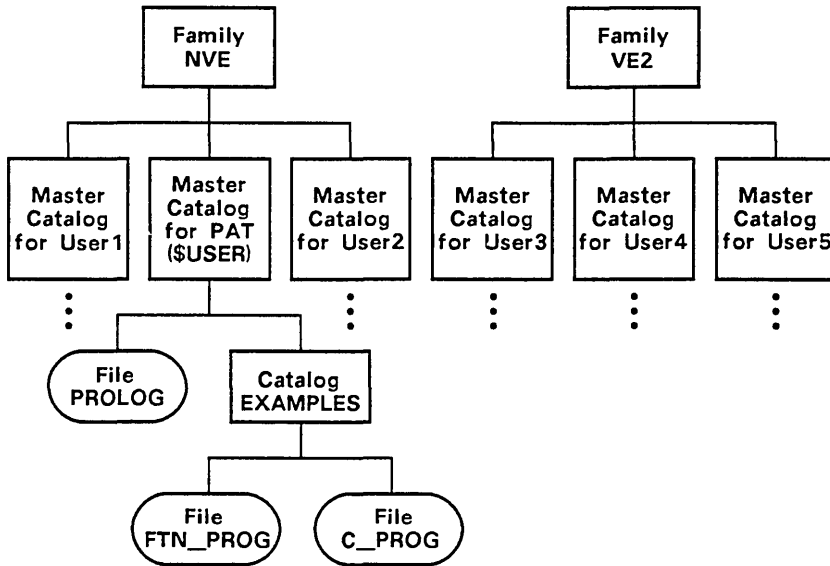
The System Command Language (SCL) is a block structured language that is both the command language for NOS/VE and a high-level programming language. That is, you can use SCL to enter system commands and to write commands of your own.

The SCL Interpreter processes all commands before submitting them to NOS/VE for execution. The SCL Interpreter examines each command for correct syntax and then evaluates any parameters specified for the command.

All NOS/VE commands, command utilities, functions, and control statements adhere to naming and syntax conventions, making NOS/VE a homogeneous system that is easy to learn and use.

The File System

Information is organized and stored on NOS/VE in files. Files, in turn, are organized and stored in catalogs. NOS/VE has a hierarchical catalog system that can be visualized in the following way:



Each NOS/VE user has a master catalog in which they can store permanent files and catalogs. NOS/VE users are members of a family of users who can share files with one another and with members of other families of NOS/VE users.

NOS/VE files can be found by tracing the path through the catalog hierarchy to the file. You reference the file by specifying the name of each catalog preceding the file before specifying the file's name.

The file system provides users the ability to set security privileges for files within their master catalog. This includes the naming of passwords for files as well as specifying the ways other users may use the files.

In addition to the master catalog and its catalogs and files, every NOS/VE user has access to a temporary and a system catalog. The temporary catalog contains files that exist for the user only for the length of each individual job. The system catalog contains files and libraries used to support NOS/VE and its products.

Working Environments

Your working environment determines the processing capabilities to which you are allowed access. Your default working environment is determined by how your site validates you to use the system. NOS/VE allows you to tailor certain aspects of your working environment to better suit your needs.

Your default working environment generally includes access to the files and libraries that support the features available at your site. These files and libraries support such NOS/VE features as the System Command Language, all of the NOS/VE commands available to you, error reporting and online documentation abilities, the programming languages (such as FORTRAN) available at your site, and a variety of other abilities available on NOS/VE.

As a part of this working environment, NOS/VE supplies a default command list to every user of NOS/VE. This command list contains every command available to you. NOS/VE also offers you the ability to tailor the command list by adding and deleting commands in the list, as well as re-ordering them. Because of this ability, you can create your own programs and commands and then add them to your working environment by adding them to the command list. In this way you can execute these commands and programs in the same way you would execute any NOS/VE command -- simply by entering its name to the SCL interpreter.

NOS/VE Attributes

A NOS/VE attribute is a characteristic of specific parts of the system. For instance, both files and jobs have attributes associated with them that describe the file or job and dictate how they can be used. Attributes allow you to control the behavior of various elements of NOS/VE by controlling the values of the attributes associated with those elements.

Networks

When you begin an interactive session, your terminal is connected to the CYBER using a collection of hardware and software that is collectively known as a network. A network transfers information back and forth between your terminal and the computer.

Networks give NOS/VE users the ability to interact with more than one computer. Through the use of networks, you can establish a connection to a computer, interrupt that connection, and then connect to another computer. You can also use networks to submit batch jobs from one computer to another, and to send files to specific devices (such as a printer) for processing.

Your site has a choice of several different networks that can be used with NOS/VE including NOS/VE's primary network, CDCNET.

Online Help

NOS/VE provides several different ways for you to obtain help while you are logged in. This help includes prompting for incorrectly entered parameter values, commands that return command and function parameter descriptions, and online manuals with more detailed information about NOS/VE.

Dual-State Systems

Some CYBERs run two different operating systems at the same time. These CYBERs are said to be dual-state systems. Every dual-state system runs NOS/VE as one of its operating systems and either a NOS or a NOS/BE operating system as the other. A NOS dual-state system, then, runs NOS and NOS/VE; a NOS/BE dual-state system runs NOS/BE and NOS/VE.

Input to SCL	2-2
Prompts	2-4
Input Lines	2-5
SCL Names	2-6
Characters You Can Use	2-6
Using Uppercase and Lowercase Letters	2-8
Examples of Valid and Invalid Names	2-8
Reserved Names	2-9
Using Spaces	2-10
Using NOS/VE Commands, Utilities, and Functions	2-11
Commands	2-11
Naming Conventions	2-11
Entering Commands	2-12
Abbreviating Commands	2-12
NOS/VE Command Lists	2-13
Command Descriptions	2-13
Parameters and Values	2-14
Specifying Parameters	2-14
Order Dependence	2-15
Abbreviating Parameters	2-16
Specifying Value Lists	2-17
Specifying a Simple Value	2-17
Specifying a Range as a Value	2-18
Specifying Value Sets	2-19
Specifying Expressions as Values	2-20
Types of Values	2-21
Using the STATUS Parameter	2-23
Command Utilities	2-24
Starting a Utility	2-24
Stopping a Utility	2-25
Functions	2-26
Calling a Function	2-27
Displaying the Value of a Function	2-28
Data Representation	2-29
Constants	2-29
Variables	2-30
SCL Types	2-31
Assignment Statements	2-32
Expressions	2-33

Control Statements	2-34
Examples of Statements	2-35
Procedures	2-36

The System Command Language (SCL) is a block-structured language that performs two basic roles:

- As a command language, SCL provides the means for submitting statements (such as compiler calls, program execution commands, and system commands) to NOS/VE. All statements are first processed by the SCL interpreter and then submitted to NOS/VE for execution. The SCL interpreter examines each statement for correct syntax and then evaluates statement parameters. If a statement contains any syntax errors or invalid parameters, or is otherwise not recognized as valid, SCL reports the errors to you and does not attempt to execute the statement.
- As a programming language, SCL supports variables, functions, expressions, and the grouping of SCL statements into procedures and utilities.

This chapter focuses on SCL as the command language for NOS/VE. It includes the following topics:

- Basic SCL syntax requirements
- NOS/VE naming conventions
- Using commands, utilities, and functions
- Specifying parameters

The last sections of this chapter also provide introductory information about SCL as a programming language. For more information about SCL as a programming language, see chapter 10, SCL Variables, Types, and Expressions; chapter 11, Command Streams and Condition Processing; and chapter 12, Writing SCL Procedures and Command Utilities.

Input to SCL

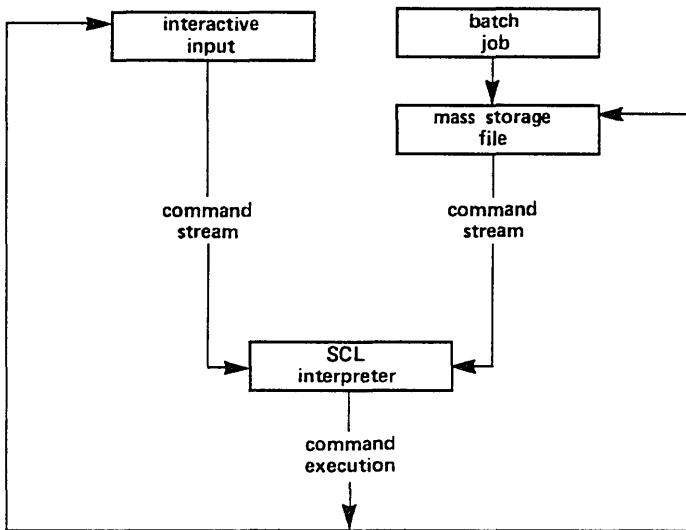
A *statement*, such as a system command or an assignment statement, is the basic unit of input interpreted by the SCL interpreter. SCL statements must meet uniform syntax requirements regarding the use of names and spaces. In addition, parameter values passed to commands and functions must match the predefined data type (such as integer or boolean) for that parameter.

The SCL interpreter processes all statements to NOS/VE. The list of statements that the SCL interpreter receives is called the *command stream*.

The source of input to the SCL interpreter varies, depending on whether you enter NOS/VE commands through an interactive terminal or whether you submit batch jobs.

- When you are at an interactive terminal, statements are processed directly by the SCL interpreter.
- When you submit a batch job, the statements are first placed in a mass storage file and then processed by the SCL interpreter.

The following figure illustrates the flow of data to the SCL interpreter:



The SCL interpreter monitors the command stream, and either submits the statements to NOS/VE for execution or returns a diagnostic message if an error is encountered.

The following paragraphs supply a brief description of SCL input requirements. For a full syntactic description of the requirements, see appendixes F and G.

Command streams are discussed in chapter 8, Command Streams and Condition Processing.

Prompts

A NOS/VE job is typically organized into several groups of SCL statements called *blocks*. At a minimum, each job has one block called the *job block*. The job block contains the SCL statements from job initiation to job termination.

During an interactive job, SCL prompts you for input in different ways, depending in part on what kind of block you are in. For example, in the job block of an interactive job, SCL prompts for input as follows:

```
/
```

This prompt indicates that NOS/VE has processed any preceding input, and is ready to accept additional input.

The system prompt (/) may be preceded by additional characters identifying a particular kind of block, such as the block created by a control statement. For example:

```
/i=0  
/loop      "Initiate a LOOP control block"  
loop/i = i + 1  
loop/display_value value = i  
loop/exit when i = 10  
loop/loopend "End the control block"  
/
```

Control statements are discussed later in this chapter.

In certain situations, the system provides options for changing the input prompt to a user-selected string. For more information, refer to the description of the PROMPT_STRING parameter of the CHANGE_CONNECTION_ATTRIBUTES command in the NOS/VE Commands and Functions manual.

Input Lines

Input lines are composed of one or more statements. The following discussion applies to all input lines, whether they are in batch jobs, interactive jobs, or procedures. The examples in the following discussion include the prompts issued for interactive input lines.

To enter more than one statement on an input line, separate the statements from one another using semicolons. The semicolon functions only as a delimiter and is not required as a terminator at the end of an input line. The following example contains two statements on a single input line:

```
/string1 = 'String 1 is a string variable.'; display_catalog
```

If an input line is longer than the width of your terminal screen or input device, you can enter it on more than one physical line.

To do so, enter an *ellipsis* (two or more consecutive periods) at the end of the line you are continuing. The system prompts for continued lines by preceding the input prompt with an ellipsis. The following single input line is contained on two physical lines:

```
/string2 = 'This is a very long string that is contin..  
../ued over two physical lines.'
```

The following single input line contains three statements on two physical lines:

```
/done = true; copy_file input=..  
../file1 output=file2; display_catalog
```

To include an ellipsis as an actual part of a line (for example, as part of a string), follow the ellipsis with a character other than a period or space.

An input line cannot exceed 65,535 characters (a line continuation ellipsis is not counted in the 65,535-character limit). SCL begins processing a continued input line only after you have entered all physical lines. It then merges the physical lines into a single input line, interprets the line's syntax, and evaluates parameters and expressions.

SCL Names

NOS/VE uses SCL names to identify command and parameter names, files, and catalogs. Whether the name is predefined by the system or one that you choose, the rules for forming it are the same. A name can be any combination of alphanumeric characters, underscores, and other special characters (listed in table 2-1) as long as it does not begin with a numeric character and contains 31 or fewer characters.

NOS/VE does not distinguish between uppercase and lowercase letters in a name: for example, it interprets the names MY_FILE and My_File as being identical.

Characters You Can Use

The table on the following page contains a list of the characters you can use when forming SCL names.

Table 2-1. Characters in SCL Names

Character	Description
a-z	Lowercase alphabetic
A-Z	Uppercase alphabetic
0-9	Numeric
_	Underline
\$	Dollar sign ¹
#	Number sign
@	Commercial at
[Opening bracket ²
]	Closing bracket ²
\	Reverse slant ²
^	Circumflex ²
`	Grave accent ²
{	Opening brace ²
}	Closing brace ²
	Vertical line ²
~	Tilde ²

1. A variable name, or file name defined by NOS/VE contains a dollar sign to distinguish it from a user-defined name. Names for SCL variables may not begin with a dollar sign. In general, avoid defining names that contain the dollar sign character in any position.

2. The special characters [,], \, ^, `, {, }, |, and ~ apply to natural languages requiring additional characters. Unless your language requires them, we recommend you avoid using these characters in names.

Using Uppercase and Lowercase Letters

The system translates lowercase letters in names to their uppercase equivalents. As a result, the following names are equivalent:

NAME_with_UPPER_and_lower_CASE
name_WITH_upper_AND_LOWER_case

Examples of Valid and Invalid Names

The following are examples of valid names:

new_data_file
#3
user_123
location@349_231
part_number_804284_472383

The following are examples of invalid names:

Invalid Name	Reason
123_3456A	Begins with a digit.
abc.345	Contains an invalid character (period).
this_name_is_going_to_be_too_long	Contains more than 31 characters.

Reserved Names

The following NOS/VE names are reserved for system use. For example, if you try to create a file with one of the following names, you will receive an error message.

- \$ASIS
- \$BOI
- \$BOP
- \$COMMAND
- \$COMMAND_OF_CALLER
- \$EOI
- \$FAMILY
- \$HIGH
- \$JOB
- \$LOCAL
- \$LOW
- \$NEXT
- \$PROC
- \$SYSTEM
- \$TASK
- \$USER
- \$UTILITY

Using Spaces

SCL recognizes the following entries as a single space:

- One or more ASCII space (SP) characters.
- One or more ASCII horizontal tab (HT) characters.
- An SCL comment.
- Any combination of the above.

For more information about ASCII characters, see the character set in appendix C. Ways of using SCL comments are discussed in chapter 9, Writing SCL Procedures and Command Utilities.

In commands, use spaces to separate parameters from the command and from each other. Spaces can also be used to improve the readability of statements in the procedures you write. For example, in the following LOGIN command, spaces delimit the command parameters and improve the readability of the command:

```
login login_user = smith password = jones
```

Since the system recognizes spaces as separators, you cannot use them within names. For example, the following forms of the LOGIN command are not valid:

Invalid Command	Reason
<pre>log in login_user=user_123 .. password=pass_456</pre>	Space appears within the command name LOGIN.
<pre>login login_user=user _123 .. password=pass_456</pre>	Space appears within the user name USER_123.
<pre>login login_user=user_123.. password=pass_456</pre>	No space before the ellipsis.

In addition, you cannot use spaces before or after operators in a parameter value (for examples, see Specifying Expressions as Values later in this chapter).

Using NOS/VE Commands, Utilities, and Functions

The following sections describe the requirements and conventions for using NOS/VE commands, utilities, functions, and their parameters.

Commands

Under NOS/VE, a *command* initiates a specific operation, such as creating or deleting a file. During an interactive session, you enter NOS/VE commands after the input prompt.

Commands consist of a command name that may be followed by one or more parameters separated by commas or spaces. Commands have the following format:

command parameter ... parameter

Naming Conventions

Each NOS/VE command follows the same format: it begins with a verb and is followed by an object which can be one or more words separated by underscores. An example is the command CHANGE_TERMINAL_ATTRIBUTES. The verb is CHANGE and the object is the two words TERMINAL_ATTRIBUTES.

The verb_object format makes SCL commands self-descriptive. Therefore, you can easily recognize the purpose of commands and anticipate the names for commands you have not yet used. For example, the DELETE_FILE command deletes a file and the DISPLAY_FILE_ATTRIBUTES command displays file attributes.

Entering Commands

NOS/VE executes commands when you enter one of the following:

- The name of a recognized command.
- The file path of an executable file.

For example, the following entry causes the system command `DISPLAY_CATALOG` to be executed:

```
/display_catalog
```

The following entry causes a procedure or a list of commands residing in file `$USER.SAMPLE_PROC` to be executed:

```
/$user.sample_proc
```

Files and file paths are discussed in chapter 4, Catalog and File Management. For more information on entering commands, see chapter 5, Command and SCL Procedure Execution.

Abbreviating Commands

You can abbreviate commands by using the first three letters of the command's verb followed by the first letter of each word in the command's object with no underscores. For example, the following commands are shown with their abbreviations:

```
CREATE_FILE or CREF
```

```
SET_WORKING_CATALOG or SETWC
```

The following are exceptions to this standard:

- The word `PASSWORD` when used in command or parameter names is abbreviated `PW`. For example, `CHANGE_LOGIN_PASSWORD` is abbreviated `CHALPW`.
- The words `MINIMUM` and `MAXIMUM` when used in command and parameter names are abbreviated `MIN` and `MAX` respectively.

NOS/VE Command Lists

A command name is recognized by the SCL interpreter if it appears as an entry in the *command list* (a list of the commands currently available for your use).

When you log in to NOS/VE, the command list initially contains entries for all the commands the system provides. This list is called the \$SYSTEM command list. You can delete any entries in the command list (including those for NOS/VE system commands), replace them with entries for commands you create, or add other entries of your own.

You can also control the order in which the command list is searched. If you have added an entry to the command list with the same name as a system command, COPY_FILE for example, you can search the command list so that one version of the command is chosen over the other.

Any changes you make to the command list apply only to your current interactive or batch job. Command lists are described in chapter 5, Command and SCL Procedure Execution.

Command Descriptions

Detailed descriptions of commands, functions, control statements, and utility subcommands provided by NOS/VE and processed by the SCL interpreter are found in the NOS/VE Commands and Functions manual. These descriptions are arranged alphabetically by name and contain:

- General information about the command
- Examples which show how to use the command
- Information about syntax requirements
- Information about parameters and their requirements

Parameters and Values

Commands often contain *parameters* that let you select various processing options. For each parameter, the command defines a value type such as integer or boolean. When you specify the value for a parameter, it must match the defined value type.

Specifying Parameters

Parameter names consist of one or more words, separated by underscores, and describe the parameter (such as OUTPUT or TERMINAL_MODEL).

You can specify a command parameter either as a parameter name equated to a list of values (called the *value list*) or as a value list alone.

```
parameter name = value list
```

or

```
value list
```

You must separate parameters from the command and from other parameters by commas or spaces. The following example calls the COPY_FILE command to copy the contents of file MY_INPUT into file MY_OUTPUT. Two parameter names and their value lists are specified.

```
/copy_file input=my_input output=my_output
```

You can enter the same command and include only the value lists as follows:

```
/copy_file my_input my_output
```

Parameter names follow the same rules as other SCL names.

Order Dependence

The order in which you enter parameters depends on whether you specify the parameter names.

- When you specify parameter names, you can list the parameters in any order, as shown in the following examples:

```
/copy_file input=my_input output=my_output
```

```
/copy_file output=my_output input=my_input
```

Each command description defines a particular order for its parameters. When you enter a parameter name, the system places the parameter in its pre-defined position.

- When you omit parameter names, you must specify the parameters *positionally*. That is, you must enter them in the order shown in the format of the command description. In the following example, the second parameter is listed without a name. Because the parameter is defined as second in the parameter list, you must enter it in that position.

```
/copy_file input=my_input my_output
```

To position to a particular parameter, include the appropriate number of commas. For example, to specify the second and fifth parameters, enter a command with the following format:

```
command, ,value list, , ,value list
```


Abbreviating Parameters

You can abbreviate parameter names by taking the first character from each word in the parameter name. For example, the abbreviation for the `COMMAND` parameter is `C`, and the abbreviation for the `OUTPUT_DESTINATION_USAGE` parameter is `ODU`.

The following are exceptions to this standard:

- `STATUS` has no abbreviation.
- `PASSWORD` is abbreviated `PW`.
- `OUTPUT_DESTINATION` is abbreviated `ODE`.
- `OUTPUT_DISPOSITION` is abbreviated `ODI`.
- The words `MINIMUM` and `MAXIMUM` when used in command and parameter names are abbreviated `MIN` and `MAX` respectively. For example, `MAXIMUM_WORKING_SET` is abbreviated `MAXWS` and `MINIMUM_WORKING_SET` is abbreviated `MINWS`.

Specifying Value Lists

A *value list* specifies one or more values that can be processed for a parameter. Each parameter defines the manner in which you can specify the values.

In SCL, you can use the following kinds of values:

- Simple value
- Range of values
- Value set

Specifying a Simple Value

A simple value is a single expression of a particular type such as string or integer. The following are simple values:

Value	Type
10	Integer
8+4	
'Mon-Fri'	String
'Hello '//there'	

To include more than one value in a value list, enclose the entire list in parentheses and delimit each value with a comma or a space.

(value,value,... value)

For example:

*(10,2*3 8-1)*

A space following the opening parenthesis or preceding the closing parenthesis is optional.

For more information about expressions, see chapter 7, SCL Variables, Types, and Expressions.

Specifying a Range as a Value

You can enter a parameter value as a *range* by showing it as two values of the same type separated by an ellipsis (two consecutive periods).

value..value

A range represents the values from the first value listed through the second value, provided the first value is less than or equal to the second.

The following are range values:

Value	Type
1..10	Integer range
false..true	Boolean range
'a'..'z'	String range

A parameter defined to accept a range of values will accept a simple value, that is, a value with a range of one. The following examples are valid range values whose value is the integer 5:

5

5..5

To include more than one range value in a value list, enclose the entire list in parentheses and delimit each range value with a comma or a space.

(value..value,value..value,...,value..value)

For example:

(1..10,0..3 -1..+1)

A space following the opening parenthesis or preceding the closing parenthesis is optional.

Specifying Value Sets

A simple value and a range of values are each considered a single *value element*. A *value set* contains two or more value elements, and is specified as follows:

(value element,value element,...,value element)

You can include simple values and value ranges in the same value set. The following are examples of value sets:

Values	Types
(21,false)	Integer, boolean
('Error code : ',700)	String, integer
(ascii,hexidecimal)	Keyword, keyword

To include more than one value set in a value list, enclose the entire list in parentheses and delimit each value set with a comma or a space.

((*value set*),(*value set*),...,(*value set*))

For example:

((0,false),(1,true) (1..3,true))

A space following the opening parenthesis or preceding the closing parenthesis is optional.

Specifying Expressions as Values

You can represent each value or value element as an expression.

The following expressions are valid parameter values:

10

(i+j)

(a*b**c)

Expressions for parameters cannot contain spaces surrounding operators.

The following are examples of invalid expressions for parameters:

Invalid Expression	Reason
<code>display_value value=3 +5</code>	Contains a space before the + operator.
<code>display_value a * b**c</code>	Contains a space before and after the * operator.

Types of Values

SCL assigns a certain *type* (such as integer or boolean) to each parameter. The SCL interpreter verifies that the value you assign to a parameter matches its system-defined type. If there is a mismatch, the SCL interpreter terminates the command and either issues a diagnostic message or prompts you for the correct type of value.

The following table lists and defines the types of parameter values recognized by SCL. Any of these values can be represented as an expression. For more information about SCL types, see chapter 7, SCL Variables, Types, and Expressions.

Type	Description
File	Reference to a file or catalog. Each element of a file reference must be a valid SCL name. The entire reference cannot exceed 256 characters including colon (:) and period (.) separators. For more information, see the description of files and file references in chapter 4.
Name	Combination of 1 through 31 alphabetic, numeric, and special characters. An SCL name you create cannot begin with a numeric character. For more information, see the discussion of SCL names in this chapter.
String	Combination of up to 256 alphabetic, numeric, and special characters enclosed in single quotation marks.
Integer	Integer value in bases 2 through 16. Numeric characters express integer values. A combination of numeric characters and the alphabetic characters A, B, C, D, E, and F (uppercase or lowercase) represents integers in bases 11 through 16.

Type	Description
Boolean	Value that is either true or false. Boolean constants are TRUE, YES, ON, FALSE, NO, and OFF.
Real	Sequence of digits containing a decimal point and an optional trailing exponent with an optional sign.
Status	Record consisting of three fields (NORMAL, CONDITION, and TEXT) that contain the completion status of a command. A brief discussion of the status parameter is presented in the following section.
Any	Any expression. Certain parameters allow for values that either are not of a specific kind or can be one of several different kinds.
Application value	Value whose syntax and meaning are defined by an application or utility.
Keyword	Name that has special meaning in the context of a particular parameter. In general, keywords are used to restrict acceptable values to a predefined set of values.

Keywords are also used in addition to one of the other value kinds for a parameter. For example, a parameter named COUNT might normally expect integer values but could be defined to accept the keyword value ALL. If you subsequently specify COUNT=ALL, the system repeats a specified operation until some limit is reached (for example, until all tape marks are skipped).

Using the STATUS Parameter

Every NOS/VE command has an optional STATUS parameter which you can use to check for error conditions.

A status variable has three fields. After processing a command, the system passes a boolean value to field NORMAL: TRUE if the command was processed correctly and FALSE if the command could not be processed. If NORMAL = FALSE, the system passes an integer code that uniquely identifies the error that occurred to a field called CONDITION. The TEXT field contains additional information about the condition.

If an abnormal condition occurs during the execution of the command, the system passes information about that condition to the status variable. You can include commands in your job to check the contents of the status variable and to change the flow of execution based on the result.

For more information about SCL variables, see chapter 7, SCL Variables, Types, and Expressions. For more information about the STATUS parameter, and condition handling, see chapter 8, SCL Command Streams and Condition Processing.

Command Utilities

Command utilities are special commands that make a set of *subcommands* available to you. The subcommands perform the particular operations of a utility. For example, the EDIT_FILE utility provides subcommands for editing files; the CREATE_OBJECT_LIBRARY utility provides subcommands for maintaining object libraries.

When you enter the command to initiate a utility, the system adds the subcommands associated with the utility to your command list. While a utility is executing, its subcommands obey the same rules and conventions as other commands. When you exit from a utility, however, its subcommands are removed from the command list and are no longer available for your use.

Starting a Utility

To start a command utility, enter the utility's command name and any associated parameters. During an interactive job, the system responds by displaying the utility's unique prompt. For example, if you start the CREATE_OBJECT_LIBRARY utility, the system responds with the COL/ prompt:

```
/create_object_library  
COL/
```

While the command utility is executing, you can enter both NOS/VE commands and the utility's subcommands. You can display a list of the subcommands and functions available within a utility by entering the `DISPLAY_COMMAND_LIST_ENTRY` command. Once you end the utility, the utility's subcommands are no longer available.

The `CREATE_OBJECT_LIBRARY` command is documented in the Object Code Management manual. The `DISPLAY_COMMAND_LIST_ENTRY` command is documented in the NOS/VE Commands and Functions manual.

Stopping a Utility

Every NOS/VE utility has a subcommand called `QUIT`. To stop a utility, enter the `QUIT` subcommand. For example, to stop the `CREATE_OBJECT_LIBRARY` utility, enter `QUIT` after the utility prompt:

```
COL/quit  
/
```

Functions

NOS/VE provides a set of *functions* that perform operations such as converting data, returning attribute information about your current environment, and accessing the system date and time. Command utilities may provide functions that perform operations specific to the utility.

NOS/VE provides two sets of functions, which are used to return values of various system attributes and variables:

- Functions used anywhere within an SCL statement.
- Functions used within procedures to determine various attributes of procedure parameters. See chapter 9, Writing SCL Procedures and Command Utilities, for further information about procedures.

Some NOS/VE utilities also provide functions. These functions are available *only* while the utility is executing. A utility's functions are described in the documentation for that utility. Use the HELP command to access online documentation for a utility.

To display available functions, use the DISPLAY_COMMAND_LIST_ENTRY command and specify FUNCTIONS as the value for the DISPLAY_OPTIONS parameter. To display the functions available for a specific utility, enter the preceding command from within that utility.

Calling a Function

To call a function, enter the name of the function, followed optionally by a list of parameters enclosed within parentheses as follows:

```
$function name(parameter list)
```

The following considerations apply when calling functions:

- All function names begin with the dollar sign character; an example of a function name is \$DATE.
- Each parameter in the list represents a value and is separated from other parameters by a space or comma as follows:


```
$function name(value)
      or
      $function name(value,value, ... value)
```
- Spaces following the opening parenthesis or preceding the closing parenthesis are optional.
- If you do not include a parameter list, you need not enter the opening and closing parentheses.
- There can be no space between the function name and the opening parenthesis of the parameter list.

The following is a call to the function that returns the current date:

```
$date
```

You can also call the function as follows:

```
$date()
```

The call to the \$DATE function is replaced with a string representing the current date. Assume, for example, that the following expression is executed on March 28, 1987:

```
'The current date is '//$date
```

The result is a string with the following value:

```
The current date is 1987-03-28
```

The \$DATE function accepts one optional parameter that specifies the format in which the date is to be returned. If you omit this parameter, a default format is used. If you specify the parameter MONTH, as in the following example:

```
$date(month)
```

the date is returned as shown in the following example:

```
March 28, 1987
```

Displaying the Value of a Function

To display the value that a function returns, use the DISPLAY_VALUE command. This command is useful both in procedure writing and during interactive sessions.

For example, to find out the current date during an interactive session, you could enter the following after the system prompt:

```
/display_value $date(month)
```

The system responds as follows:

```
March 28, 1987
```

```
/
```

Data Representation

Each NOS/VE job consists of data and statements that perform some operation on the data. This data can be contained within files; data can also be represented by constants and variables.

Constants

A *constant* specifies a fixed data value to the SCL interpreter. You can use constants anywhere within jobs, procedures, or utilities.

NOS/VE provides several functions that return constant values; these functions are particularly useful to procedure writers. For example, the \$MAX_INTEGER function returns the largest positive integer for the system.

The following is a list of functions which return constant values:

\$MAX_INTEGER	Returns the maximum positive integer allowed for a parameter (9,223,372,036,854,775,807).
\$MAX_NAME	Returns the maximum length of a name allowed for a parameter (31).
\$MAX_STRING	Returns the maximum length of a string allowed for a parameter (256).
\$MAX_VALUE_SETS	Returns the maximum number of a value sets allowed for a parameter (2,147,483,647).
\$MAX_VALUES	Returns the maximum number of values allowed per value set (2,147,483,647).
\$MIN_INTEGER	Returns the minimum integer value allowed for a parameter (-9,223,372,036,854,775,808).

Variables

A *variable* is a name that refers to a data item whose value can be changed during the execution of a job. You can use variables anywhere within a job, but they are generally known only within the block in which they are created.

An SCL variable must be assigned a data type when it is created. If you create a variable explicitly with the `CREATE_VARIABLE` command, you specify the variable's type using the `KIND` parameter. If you create a variable implicitly, you use an assignment statement to assign a value to a previously undefined variable. The variable inherits the data type of the expression to which it is equated.

SCL Types

The following table shows the most common data types and examples of values for each type:

Type	Example
Name or Keyword	N2000 ALL
Integer	100 -5000
String	"This is a string." 'Names don't have apostrophes, but strings do.' ¹
Boolean	TRUE or FALSE YES or NO ON or OFF
File	MY_FILE \$USER.MY_FILE \$LOCAL.MY_FILE

For more information about variables and types, see chapter 7, SCL Variables, Types, and Expressions.

1. To include an apostrophe within a string, use two consecutive apostrophes

Assignment Statements

Assignment statements assign values to variables. The general form of the assignment statement is as follows:

`variable = expression`

You can implicitly create variables by assigning them a value in an assignment statement, and you can explicitly create or delete variables using commands. If you explicitly create a variable, you specify its data type (for example, integer or boolean). If you implicitly create a variable, its data type is the same as the expression to which it is equated.

The following are examples of assignment statements:

<u>Assignment Statement</u>	<u>Data Type</u>
<code>factor = 10*j</code>	Integer
<code>file_deleted = true</code>	Boolean
<code>first_name = 'John'</code> <code>last_name = 'Doe'</code> <code>full_name = last_name//', '//first_name</code>	String

To assign a new value to a variable that already exists, the value of the expression on the right side of the equals sign must be of the same type as the variable on the left.

Expressions

An *expression* is the representation of a single value. An expression can be a constant or a variable appearing alone, or constants, variables, and operators combined to compute a value.

The following considerations apply to expressions:

- An expression can contain any number of data items, but after it is evaluated, it represents a single value.
- If an operation is performed on two data items of differing data types, an error occurs. NOS/VE provides a number of functions which convert data items of one data type to items of another type.
- When the system encounters a NOS/VE-defined constant or function in an expression, it evaluates the constant or function (and any supplied arguments) and uses the resulting value in the expression.
- In an expression, the system evaluates operands within parentheses first, with the innermost parentheses taking precedence.
- In an expression used in an assignment statement, you can surround operators with spaces; in an expression used in a parameter, you cannot enter spaces surrounding the operators (for examples, see *Specifying Expressions as Values* earlier in this chapter).

You can use expressions for any parameter of a NOS/VE command, command utility, function, or control statement. Expressions are also valid for any parameter of an SCL procedure. Rules for expressions are described in chapter 7, *SCL Types, Variables, and Expressions*.

Control Statements

A *control statement* structures and controls the flow of a command stream. With control statements, you can structure a command stream into blocks of statements called statement lists. These statement lists can be executed until either a required condition occurs, a repetition count is reached, or a specific exit is requested from the block.

SCL provides the following types of control statements:

- *Structured statements* used to structure a command stream into groups of statements either processed as separate entities or until a specified terminating condition or statement is encountered.
- *Flow control statements* used to control the flow of a structured statement.
- *Conditional execution statements* used to control the execution of statements based on the evaluation of an associated boolean expression.
- *Condition handler statements* used to execute groups of statements when specified abnormal conditions occur.

Control statements are described in chapter 8, SCL Command Streams and Condition Processing.

Examples of Statements

The following example contains assignment statements, a command, and a control statement.

<code>i=10</code>	Assignment statement.
<code>for j = 1 to i by 1+1 do</code>	FOR control statement begin clause.
<code>k=j*j</code>	Assignment statement.
<code>display_value k</code>	DISPLAY_VALUE command.
<code>forend</code>	FOR control statement end clause.

In the example, the following activity takes place:

1. The variable I is created (variables can be referenced in either uppercase or lowercase) and assigned the initial value 10. The variable I is used as an iteration limit for the subsequent FOR control statement.
2. The variable J is implicitly created by the FOR statement. J is given the initial value of 1 and is incremented by 2, that is 1+1, each time the FOR statement is executed.
3. The variable K is implicitly created and then assigned the value of J times J.
4. The DISPLAY_VALUE command is used to display the value of k.

Execution of the preceding statements produces these results:

```

1
9
25
49
81

```

Procedures

A *procedure* is a list of statements that are processed like a single command when the procedure is called by its name. The syntax of a procedure call is identical to the syntax of NOS/VE commands, including an optional list of parameters to which values can be passed. It is therefore possible to use procedures both to create your own commands and to replace NOS/VE commands with your own versions. Procedures are described in chapter 9, Writing SCL Procedures and Command Utilities. Altering the NOS/VE command list is described in chapter 5, Command and SCL Procedure Execution.

Interactive Sessions

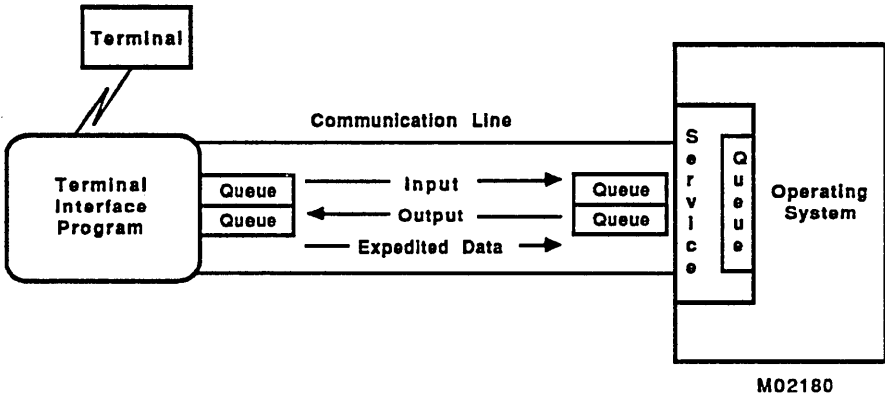
3

Networks	3-2
Logging In and Out	3-4
Login Information	3-5
Changing Your Password	3-7
Entering Commands Interactively	3-9
Line and Screen Mode Processing	3-9
Using Screen Mode Processing	3-10
Network Command Characters	3-12
User-Controlled Interrupts	3-13
Pause Break	3-15
Terminate Break	3-16
Screen-Mode Pause and Terminate Breaks	3-16
NAM/CCP Processing	3-17
NAM/CDCNET Processing	3-17
NAMVE/CDCNET Processing	3-18
Terminal Disconnects	3-19
Reestablishing Terminal Connections	3-19
Job Time Limits	3-20
Displaying Your User Validation	3-21
Changing Your User Validations	3-23
Getting Help On Line	3-24
Online Help Commands	3-24
Using Parameter Prompting	3-26
Using Online Manuals	3-28
Summary of Screen Mode Operations	3-28
Summary of Line Mode Subcommands	3-30
The Online Examples Manual	3-31

A NOS/VE job that you manage directly from a terminal is called an *interactive session*. In order to start an interactive session, you must log in to NOS/VE. How you log in to NOS/VE and manage your job depends in part upon the type of network you are using to connect to the computer. This chapter describes some of the networks you might use to connect to NOS/VE, and then tells you how to manage a NOS/VE interactive session.

Networks

Networks are collections of hardware and software connecting your terminal to the computer. The following figure illustrates the parts of a network:



In the preceding illustration, the terminal is connected to a *Terminal Interface Program* (TIP). The TIP runs on the network's hardware and processes the data that passes between the terminal and the network's communication line.

The *communication line* connects the TIP with the host computer. The communication line has three paths: the input, output, and expedited paths. Note that the input and output paths both have queues on either end. These queues are used to store information waiting to be processed by the network. The following describes each of the communication paths:

Input path

Contains information being sent from the terminal to the host computer.

Output path

Contains information being sent from the host computer to the terminal.

Expedited path

Contains data that bypass the network's queues and is sent immediately to the host computer for processing. The expedited path is used for data which informs the operating system and/or the network, to take some kind of special action.

The communication line is connected to a service. The service is a program running on the host computer that handles the flow of information between the network and the computer.

Logging In and Out

Logging in to NOS/VE involves two operations:

1. Establishing a communications link with the host computer.
2. Identifying yourself to NOS/VE.

Both processes are easy to do, but vary depending upon the terminal you are using, the way that your terminal is connected to the network, the actual network that you are using, and whether you are using a dual-state system.

You can log in to NOS/VE through the following standard networks:

- NAMVE/CDCNET
- NAM/CDCNET
- NAM/CCP
- INTERCOM

To log out of NOS/VE, at the system prompt simply enter:

```
/logout
```

The Introduction to NOS/VE manual describes how to log in and out of NOS/VE using the standard networks. That manual, however, assumes that you have already connected your terminal to the network. If you have any questions about logging in that are not discussed in that manual, consult your site personnel.

Login Information

Before you can log in to NOS/VE, you must be validated to use the system by your site personnel. After you have been validated, you should be provided with the following information:

- **Family name**
The name of the group of users under which your permanent catalogs are located.
- **User name**
This name, combined with your family name, identifies you as a valid NOS/VE user and gives the location of your permanent files within your family.
- **Password**
This word prevents others from logging in to the system under your user name. You will use the assigned password the first time you log in to the system. At that time, you should change the assigned password to something that is known only to you. To change your password, use the `CHANGE_LOGIN_PASSWORD` command.
- **Account and project names**
These names may not be required by your site. Account and project names are used to identify who should be billed for your use of the system.
- **Network used to access NOS/VE**
Along with this information should be any network-dependent information you will need to use NOS/VE.
- **Additional instructions specific to your site, if any.**

When you log in, at a minimum the system will check the validity of your family name, user name, and password. In addition, depending upon your validation level, the system may also check the validity of the account and project information supplied to it. The following table defines the three different validation levels.

Level	Definition
USER	System checks the validity of the family name, user name, and password information supplied to it during login.
ACCOUNT	In addition to the USER level information, the system also checks the validity of the account name supplied to it during login.
PROJECT	In addition to the USER and ACCOUNT level information, the system also checks the validity of the project name supplied to it during login.

You can find out what your validation level is by using the `$VALIDATION_LEVEL` function. For instance:

```

/display_value $validation_level
USER
/

```

Note that in some cases, you can let your family name, account, and project information default to given values during login. For instance, you do not need to enter your family name during login if you are logging in to a system that has just one family on it. Similarly, for ACCOUNT and PROJECT validation levels, you need not specify account and project information during login if the appropriate values are set up for your user validation. See *Displaying Your User Validation* later in this chapter for information on your user validation. See the *Introduction to NOS/VE* for more information about logging in.

Changing Your Password

The first time you log in to NOS/VE, you should change the password given to you by your site to a password only you know. This is done using the `CHANGE_LOGIN_PASSWORD` command. For instance, if your site assigned you a password of `A678Y59`, you can change it as follows:

```
/change_login_password old_password=A678Y59 ..
../new_password=goofy
```

Your new password is now `GOOFY`.

For security reasons, your site may choose to force you to change your password from time to time. If this is the case, when you login you will see a message displayed on your screen informing you of this. Your site determines when you will start seeing this message as well as how often you must change your password.

The number of days that your site allows to pass before you must change your password is known as the *maximum expiration interval*. The date that your password will expire is known as the *expiration date*. The *expiration warning interval* indicates when you will start seeing the message informing you that your password will expire. This value is in terms of days before your password expires.

For instance, suppose your expiration date is June 3, 1988, your maximum expiration interval is 30 days, and your expiration warning interval is 10 days. Then starting on May 25, you will see the following message every time you log in until you change your password:

```
--INFORMATIVE CL 7018-- Your password will expire at 00:00:00
on 1988-06-03.
```

If you do not change your password by midnight, June 2, 1988, your password will expire and you will no longer be able to log in.

If you change your password on June 2, then your new expiration date will be 30 days from then, or at midnight on July 1.

You can use the `CHANGE_LOGIN_PASSWORD` command to set your own expiration interval and expiration date. To do this, use either the `EXPIRATION_DATE` parameter to explicitly set an expiration date, or use the `EXPIRATION_INTERVAL` parameter to set an interval that the system will use to calculate the new expiration date. If you specify both parameters on the call to `CHANGE_LOGIN_PASSWORD`, the system will use the date specified on `EXPIRATION_DATE` as the new expiration date, and will use the value specified on `EXPIRATION_INTERVAL` to calculate future expiration dates.

Note that the values you specify on the `EXPIRATION_DATE` and `EXPIRATION_INTERVAL` parameters may not exceed the maximum expiration interval permitted to you by your site.

You can find out what your expiration interval, expiration date, expiration warning interval, and maximum expiration interval are by displaying your user validation using the `ADMINISTER_VALIDATION` utility. This process is discussed in *Displaying Your User Validation* later in this chapter.

A value of `UNLIMITED` for either the maximum expiration interval or the expiration interval values means that no time period exists in which you have to change your password. A value of `NONE` for expiration date means that your password will never expire. A value of zero (0) for expiration warning interval means that you will not be warned that your password is going to expire.

Entering Commands Interactively

Once you have started an interactive session, you manage the session by entering input lines to the system. Input lines may contain commands for the system to execute, or data for processing by programs.

You can use any of the ASCII 128-character set in the input line. You terminate a command line using the end-of-line character defined for your terminal. By default, this character is the carriage return.

By entering commands to NOS/VE, you can give instructions to the SCL interpreter, a program, a command utility, or directly to the network. You can interactively enter commands to the SCL interpreter or to a program only when you are prompted to do so by the interpreter or the program. You can usually enter commands to the network at any time.

Normally, as each input line is completed it is given to the SCL interpreter or program to process. However, if you enter input at a rate that exceeds the rate at which the system can process it, the information is saved by the system and the typed-ahead input is delivered to the interpreter or program when requested.

Line and Screen Mode Processing

NOS/VE allows interaction in both *line* and *screen* mode. When you first start an interactive session, by default your interactive job is set to line mode. You interact with NOS/VE in line mode by entering commands line by line after the system prompt (/). At this time, the INPUT_EDITING_MODE connection attribute is set to NORMAL.

Some applications, however, allow you to interact with NOS/VE in screen mode. In screen mode, you enter and edit data by moving the cursor to different positions on the screen, and the system displays messages in different areas of the screen. Note that applications using screen mode will set the INPUT_EDITING_MODE connection attribute to TRANSPARENT.

For more information on connection attributes, see chapter 13, Terminal Management.

Using Screen Mode Processing

To use screen mode, your terminal must support screen-mode interaction. Further, you must identify your terminal model and specify that you want to use screen mode interaction. The following example sets the `TERMINAL_MODEL` terminal attribute and specifies screen-mode interaction:

```
/change_terminal_attribute terminal_model=pc_connect_13
/change_interaction_style mode=screen
```

The following table lists the most commonly used terminal models. For a complete list of the NOS/VE supported terminal models, see the `TERMINAL_MODEL` terminal attribute description in chapter 13, Terminal Management.

NOS/VE Model Name	Terminal Model
CDC_721	CDC 721
CDC_722	CDC 722
CDC_722_30	CDC 722_30
DEC_VT100	DEC VT100 (18 function key definition)
DEC_VT100_GOLD	DEC VT100 (32 function key definition)
DEC_VT220	DEC VT220
MAC_CONNECT_10	Apple Macintosh (CONNECT 1.0)
MAC_CONNECT_11	Apple Macintosh (CONNECT 1.1)
MAC_CONNECT_20	Apple Macintosh (CONNECT 2.0)

NOS/VE Model Name	Terminal Model
PC_CONNECT_10	IBM PC (CONNECT 1.0)
PC_CONNECT_11	IBM PC (CONNECT 1.1)
PC_CONNECT_12	IBM PC (CONNECT 1.2)
PC_CONNECT_13	IBM PC (CONNECT 1.3)
ZEN_Z19	Zenith Z19 or Heathkit H19
ZEN_Z29	Zenith Z29

In addition to the NOS/VE-supplied terminal definitions, most sites provide other terminal definitions. To see a list of available terminal definitions, enter the following command:

```
/display_object_library $system.tdu.terminal_definitions
```

The system displays a list of terminal definitions that includes the date and time the definitions were created. Definition names consist of the terminal model name prefixed by CSM\$.

To use a terminal definition, enter the terminal model name (without the CSM\$ prefix) as the value for the `TERMINAL_MODEL` parameter on the `CHANGE_TERMINAL_ATTRIBUTES` command.

For more information about terminal definitions, refer to the NOS/VE Terminal Definition manual.

Network Command Characters

In some cases, you may also enter commands for the network to process by entering the appropriate network keystroke sequence. This is only possible if your network supports network command characters and if the `INPUT_EDITING_MODE` connection attribute is set to `NORMAL`. See `Screen Mode Pause and Terminate Breaks` for information on the network command characters you can use when `INPUT_EDITING_MODE` is set to `TRANSPARENT`.

A *network command character* is a character that indicates that the data appended to the character is to be handled in some special way by the network. The following is an example of this:

%1

In the above example, the network keystroke sequence consists of the percent sign (%) which is the network command character followed by the number one (1) which has significant meaning to the network.

To find out what your network command character is, use the `DISPLAY_TERMINAL_ATTRIBUTES` (`DISTA`) command. You may also change your current network command character to any one-character string value using the `CHANGE_TERMINAL_ATTRIBUTES` (`CHATA`) command. For example:

```
/display_terminal_attributes ..
../do=network_command_character
Network_Command_Character      : $CHAR(37)  "%"
/change_terminal_attributes ..
../network_command_character = '@'
/dista ncc
Network_Command_Character      : $CHAR(64)  "@"
```

User-Controlled Interrupts

It is often possible for you to interrupt the system's current operation and get it to perform a specific function. This is done by entering the appropriate network keystroke sequence. When you do this, you are executing a *user-controlled interrupt*. User-controlled interrupts are sent to the operating system as expedited data.

Each network has its own set of user-controlled interrupts that you may use. The following table lists the user-controlled interrupts available for each of the standard networks. Some of the following interrupts initiate processing conditions. Others provide the equivalent of a NOS/VE command.

If you are not using a standard network, consult your network's documentation or your site personnel for information on the supported user controlled interrupts.

Table 3-1. Summary of User Controlled Interrupts

Condition or SCL Command	CDCNET ¹	NAM/CCP ²	INTERCOM
Pause Break	%1	CTRL-P	%A
Terminate Break	%2	CTRL-T	%A, TERC
User Exit (LOGOUT)	%X	ESC X	none
Discard Typed-Ahead Input	%T	none	none

(Continued)

1. The first character in the sequence is the character specified by the NETWORK_COMMAND_CHARACTER attribute. Normally, this character is the percent sign (%). CDCNET refers to both NAMVE/CDCNET and NAM/CDCNET.

2. The first character in the sequence is the character specified by the NETWORK_COMMAND_CHARACTER attribute. Typically, this character is the escape character (ESC). VT100s, however, use the percent (%) character by default.

Table 3-1. Summary of User Controlled Interrupts (Continued)

Condition or SCL Command			
DETACH_JOB	%D	ESC D	%A, DETJ
DISPLAY_JOB_STATUS	%S	ESC S	%A, DISJS, RESC
DISPLAY_JOB_STATUS N=ALL	%J	ESC J	%A, DISJS N= ALL, RESC
DISPLAY_LOG DO=10	%L	ESC L	%A, DISL 10, RESC
DISPLAY_ACTIVE_ TASKS	%A	ESC A	%A, DISAT, RESC

While most of the interrupts available to you will not affect data on the network, the pause and terminate break interrupts will. The pause break and terminate break interrupts are discussed later in this section.

By default, you can use user-controlled interrupts only when the INPUT_EDITING_MODE connection attribute is set to NORMAL. However, most networks will allow you to set up the capability to use either the pause or the terminate break interrupts when INPUT_EDITING_MODE is set to TRANSPARENT.

Pause Break

The *pause break interrupt* is a condition that suspends the execution of a command. A pause break is handled in the following manner:

- Any unread typed-ahead data that currently exists on either the operating system or the network is discarded.
- The operating system suspends command execution.

A pause break is usually ignored during permanent file transfers. However, on INTERCOM a pause break is treated as a terminate break -- that is, the file transfer is terminated rather than suspended.

To suspend the execution of a command while in line mode, enter the pause break keystroke sequence for the network you are using. Once you have entered this keystroke sequence, the system responds with the following message and prompt:

```
*Suspended - 1*
p/
```

The displayed number, which identifies the pause break level, increases by 1 each time you enter a pause break without resuming or terminating the previously suspended command.

To continue execution of the suspended command, use the RESUME_COMMAND command. To terminate the suspended command, use the TERMINATE_COMMAND command. RESUME_COMMAND and TERMINATE_COMMAND operate on the most recently suspended command.

Terminate Break

A *terminate break interrupt* is a condition that terminates the execution of the current command. A terminate break is handled in the following manner:

- Any unread typed-ahead data that currently exists on either the operating system or the network is discarded.
- Any unwritten output generated by the terminated command that currently exists on either the operating system or the network is discarded.
- The operating system terminates command execution.

To terminate the execution of a command while in line mode, enter the terminate break keystroke sequence for the network you are using. After you have entered the terminate break sequence, the system responds with the following message and prompt:

```
Command terminated.  
/
```

Screen-Mode Pause and Terminate Breaks

The pause and terminate breaks are not automatically supported when you are using a utility in screen mode. This is because when you are using a utility in screen mode, the `INPUT_EDITING_MODE` connection attribute is set to `TRANSPARENT` so the network will not interpret the network keystroke sequences. However, for most networks you may establish a way to use either the pause or the terminate break interrupts when the `INPUT_EDITING_MODE` connection attribute is set to `TRANSPARENT`.

How you establish a method of controlling the processing when `INPUT_EDITING_MODE` is `TRANSPARENT`, depends on which network you are using. The following sections describe how to establish control processing when using the `NAM/CCP`, `NAM/CDCNET`, and `NAMVE/CDCNET` networks. If you are not using one of these networks, consult your network's documentation or your site personnel for more information.

Once you establish a method of controlling processing, you can use this method when `INPUT_EDITING_MODE` is `NORMAL` (as is typically the case when you are processing commands in line mode) as well as when `INPUT_EDITING_MODE` is `TRANSPARENT`.

NAM/CCP Processing

To use `CTRL-T` to terminate processing and `CTRL-P` to interrupt processing when `INPUT_EDITING_MODE` is `TRANSPARENT`, enter the `CHANGE_TERMINAL_ATTRIBUTES` command with the `ATTENTION_CHARACTER` parameter set to a non-null value. For example, enter:

```
change_terminal_attributes attention_character=$char(1)
```

NAM/CDCNET Processing

To use a user-defined character to either terminate or interrupt processing do the following:

1. Using the `CDCNET` command, `CHANGE_TERMINAL_ATTRIBUTES`, define an attention character.³
2. Using the `CDCNET` command, `CHANGE_CONNECTION_ATTRIBUTES`, set the attention character action to either:
 - 1 For the pause break condition (interrupt processing).
 - 2 For the terminate break condition (terminate processing).

3. The character you choose for the attention character must come from a subset of the ASCII character set as described in the `CDCNET Terminal Interface` manual.

For example, to define CTRL-C as the pause break character, enter:

```
%change_terminal_attributes attention_character=3
%change_connection_attributes attention_character_action=1
```

The CDCNET commands CHANGE_TERMINAL_ATTRIBUTES and CHANGE_CONNECTION_ATTRIBUTES are described briefly in the CDCNET_ACCESS online manual. They are described in detail in the CDCNET Terminal Interface Usage manual.

NAMVE/CDCNET Processing

To use a user-defined character to either terminate or interrupt processing do the following:

1. Using the NOS/VE command, CHANGE_TERMINAL_ATTRIBUTES, define an attention character.
2. Using the NOS/VE command CHANGE_TERM_CONN_DEFAULTS, set the attention character action to either:
 - 1 For the pause break condition (interrupt processing).
 - 2 For the terminate break condition (terminate processing).
3. Set the ATTENTION_CHARACTER_ACTION connection attribute for terminal files INPUT, OUTPUT, and COMMAND to the same value as the attention character action in step 2. Use the NOS/VE command CHANGE_CONNECTION_ATTRIBUTE to do this.

For example to define CONTROL/C as the pause break character, enter:

```
change_terminal_attributes attention_character=$char(3(8))
change_term_conn_defaults attention_character_action=1
change_connection_attributes terminal_file_name=input aca=1
change_connection_attributes terminal_file_name=output aca=1
change_connection_attributes terminal_file_name=command aca=1
```

Terminal Disconnects

A *terminal disconnect* is a condition that occurs when the line between the terminal and the system is disconnected or when you explicitly detach your terminal from the current interactive job.

When a terminal is disconnected, the system does the following:

- For most networks, the system discards typed-ahead input that has not been read. For NAM/VE, however, the disconnect condition will not occur until all typed-ahead input has been read by the job.
- The system continues executing the job until terminal input or output is attempted. At that time, the job is suspended and retained by the system for a period of time determined by the `DETACHED_JOB_WAIT_TIME` job attribute. For information about job attributes, see the Job Management chapter later in this manual.

To explicitly disconnect your terminal from the current interactive job, either use the user controlled interrupt or use the `DETACH_JOB` command. When explicit disconnection occurs, the system automatically initiates a new interactive job.

Reestablishing Terminal Connections

You can reconnect your terminal to a disconnected interactive job. To do so, you must reconnect your terminal to the system within the period of time allotted by the `DETACHED_JOB_WAIT_TIME` job attribute. If you do not reconnect within this period of time, the job is terminated.

Once you have reconnected your terminal to the system, you may reconnect your terminal to the suspended job using the `ATTACH_JOB` command. After reconnecting to the job, the system leaves it in a pause break condition. At this time, you may either resume or terminate the suspended command.

You cannot attach a job that originated on a different network from the one you are currently using. For example, if you were using CDCNET and your job was disconnected, and you log back in to the system using INTERCOM, you will not be able to reconnect the suspended job. To reconnect the job, log in to the system using CDCNET.

Job Time Limits

A *time limit* is a condition that occurs in a job when the central processing time resource limit for a job is reached. When this happens, the system does the following:

- Suspends all of your current activity in the job.
- Displays a message informing you that the limit has been reached.

For interactive jobs, the system will also give you the following message prompting you for an integer value to be added to your CPU time resource limit. You may either enter a value at the prompt and the system will continue executing your job, or you may logout.

```
JOB TIME LIMIT REACHED
CP_TIME resource limit has been reached.
Accumulator value is 10000.
Abort limit is 140737488.
Please enter an integer increment or LOGOUT.
?
```

Note that the increment you specify added to the accumulator value cannot exceed the abort limit. Limits are discussed in the Job Management chapter later in this manual.

Displaying Your User Validation

To display your user validation information, follow these steps:

1. Enter the ADMINISTER_VALIDATION utility.
2. At the utility prompt, enter the DISPLAY_USER subcommand.
3. To exit the ADMINISTER_VALIDATION utility, enter the QUIT command at the utility prompt.

For example:

```

/administer_validation
AV/display_user

SARETT
  CAPABILITIES
    Value: (EXPLICIT_REMOTE_FILE ..
            IMPLICIT_REMOTE_FILE ..
            MAIL_VE_LOCAL_MULTI_HOST_ACCESS ..
            MAIL_VE_SELF_ADMINISTRATION ..
            READ_UNLABELLED_TAPES ..
            TIMESHARING ..
            WRITE_UNLABELLED_TAPES)

  CPU_TIME_LIMIT
    Job warning limit:  UNLIMITED
    Job maximum limit:  UNLIMITED

  CREATION_ACCOUNT_PROJECT
    Not authorized to display value.

  DEFAULT_ACCOUNT_PROJECT
    Account: D1257
    Project: P83A2821

  JOB_CLASS
    Job classes: (BATCH ..
                 INTERACTIVE ..
                 SYSTEM_DEFAULT ..
                 EXPRESS_BATCH ..
                 FILE_TRANSFER)

    Interactive default: INTERACTIVE
    Batch default: BATCH

  LINK_ATTRIBUTE_CHARGE
    Value: ''

  LINK_ATTRIBUTE_FAMILY
    Value: 'CLSH990'

  LINK_ATTRIBUTE_PASSWORD

```

Displaying Your User Validation

Not authorized to display value.
LINK_ATTRIBUTE_PROJECT
Value: ''
LINK_ATTRIBUTE_USER
Value: 'SARETT'
LOGIN_PASSWORD
Expiration date: 1988-03-14.10:30:28
Expiration interval: 35 days
Maximum expiration interval: Unlimited
Expiration warning interval: 14 days
Password attributes: NONE
MAILVE_ADMINISTRATION
Value: SELF
MAILVE_DISTRIBUTION_LIST_LIMIT
Value: 25
MAILVE_MAILBOX_LIMIT
Value: 1
MAILVE_RETENTION_LIMIT
Value: 14
PERMANENT_FILE_SPACE_LIMIT
Total limit: UNLIMITED
Total accumulation: 0
RING_PRIVILEGES
Minimum ring: 11
Nominal ring: 11
SRU_LIMIT
Job warning limit: 50000
Job maximum limit: UNLIMITED
TASK_LIMIT
Job warning limit: 30
Job maximum limit: 30
USERS_NAME
Value: 'SARETTE S'
USER_EPILOG
Value: \$USER.EPILOG
USER_PROLOG
Value: \$USER.PROLOG
AV/quit
/

Changing Your User Validations

You may change some of your validation information using the ADMINISTER_VALIDATION utility. To do this, enter the ADMINISTER_VALIDATION utility and then the CHANGE_USER utility. For instance, it is possible for you to change the default for your account and/or project login information through the use of these utilities. Thus, if you want your default account number during login to be A668Y9 enter the following:

```
/administer_validation
ADMV/change_user
CHAU/change_default_account_project account=A668Y9
CHAU/quit
ADMV/quit
/
```

You may also change your login information for dual-state access and your prolog and epilog names using these utilities. For information on changing your login information for dual-state access, see Dual-State File Access later in this manual. For information on changing your prolog and epilog names, see Job Management later in this manual.

Getting Help On Line

NOS/VE provides the ability to display information about commands, subcommands, functions, and messages through commands and parameter prompting. This section discusses these methods of obtaining online help.

Online Help Commands

The following table summarizes the commands used display information about commands, subcommands, functions, and messages.

Table 3-2. Summary of Online Help Commands

Command	Description
DISPLAY_COMMAND_INFORMATION	Lists a command's parameters and their abbreviations, types, and default values.
DISPLAY_FUNCTION_INFORMATION	Lists a function's parameters and their types, and default values.

(Continued)

Table 3-2. Summary of Online Help Commands *(Continued)*

Command	Description
DISPLAY_COMMAND_LIST_ENTRY	Displays a list of all available commands and functions. When used within a utility, displays a list of all available subcommands and functions.
HELP	<p>Uses the EXPLAIN utility to access the help text provided in the system. The forms of help text available can be categorized either as screens from an online manual or, if you are in a utility that provides other help, as windows of help text that overlay what is on your screen.</p> <p>If you access an online manual, the manual can contain general information about a specified subject, an explanation of an error message, or information specific to the the utility you are currently using. The name of the manual always resides in the upper left hand corner of the screen.</p>

Using Parameter Prompting

In addition to the commands described in table 3-2, NOS/VE conditionally provides further online help when you call a command through parameter prompting. Parameter prompting may be enabled and disabled using the `CHANGE_SCL_OPTION` command. By default, parameter prompting is enabled.

Parameter prompting occurs when you do one of the following:

- You enter a command with one or more parameters supplied, and either one or more parameters are in error or a required parameter is missing. Errors are reported and corrections are prompted for until you have supplied all required parameters and all supplied parameters are satisfactory.
- You enter a command without specifying any parameter values, but the command has at least one required parameter. In this case, all the parameters on the command are prompted for, with the exception of the status parameter.
- You enter a question mark followed by a command reference. You can use this technique either at the NOS/VE prompt or from within an SCL procedure. The system then prompts you for all the command's parameters, whether required or not.

To stop prompting and terminate the command, enter a terminate break at the parameter prompt. If you do not enter a terminate break at the parameter prompt, the system prompts you for the remaining parameter values and executes the command, using the values you have supply for the parameters. In addition, you may enter the following:

- | | |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Carriage return | Indicates you want to use the default value for an optional parameter. |
| Question mark (?) | Obtains information about acceptable responses to the prompt. Entering a question mark a second time in response to a prompt will result in the display of information about all the command's parameters. |
| Semicolon (;) | Indicates that no more prompting should occur and the command should be executed. The semicolon can be entered as the entire response, or it can be entered at the end of a parameter response. The semicolon will work only if you have supplied all the required parameters. |
| Slant (/) | Indicates that the rest of the response is to be interpreted as a command line. The slant must be entered as the first character of the response. |

Using Online Manuals

You can access manuals on line by using the EXPLAIN utility. The EXPLAIN utility can operate in either line or screen mode. If you enter EXPLAIN with no parameters, the system displays the NOS/VE System Information Manual main menu. From this menu, you can get a list of online manuals as well as information about using online manuals.

Summary of Screen Mode Operations

If you are using EXPLAIN in screen mode, you can perform the following operations by using the corresponding function keys or dedicated keys:

Table 3-3. Summary of Screen Mode Operations

Operation	Description
RETURN or Fwd	Page forward one screen.
Bkw	Page backward one screen.
Up	Display the previous menu screen.
Find	Find information about a topic.
Index	Display a menu of topics described in the manual.
Back	Return to where you last used the Find operation or made a menu selection.
First	Display the main menu.

(Continued)

Table 3-3. Summary of Screen Mode Operations *(Continued)*

Operation	Description
Help	Display operations and help instructions. Pressed again, displays the Help menu. Pressed while cursor is positioned on function key label, displays a brief description of the function.
Copy	Copy the current screen to the file specified by the LIST parameter on the HELP or EXPLAIN command.
SetLog	Toggles logging operation on and off. When logging is on, copies the current and all subsequently displayed screens to the file specified by the LIST parameter on the HELP or EXPLAIN command.
Refresh	Redisplay the current screen.
Revert	Return to the previous manual.
Quit	Leave the online manual.
KeyOn	Turn the operations display on.
KeyOff	Turn the operations display off.

Summary of Line Mode Subcommands

If you are using EXPLAIN in line mode, you can use the following subcommands to perform operations:

Table 3-4. Summary of Line Mode Subcommands

Subcommand or key	Description
+ or RETURN	Page forward one screen.
-	Page backward one screen.
UP	Display the previous screen.
xxxx?	Find information about xxxx.
?	Find next information about xxxx.
DISI	Display an alphabetical menu of topics described in the manual.
<	Return to where you last typed xxxx? or made a menu selection.
TOP	Display the main menu.
HELP	Display the Help menu.
SETLO ON	Copy the current and subsequent screens of text to the file specified by the LIST parameter on the EXPLAIN command.
SETLO OFF	Stop copying.
CLEAR	Redisplay the current screen.
REVERT	Return to the previous online manual.
QUIT	Leave the online manual.

The Online Examples Manual

The Examples manual is an online menu-driven manual containing fully executable examples. These examples reflect concepts found in various areas of NOS/VE such as:

- COBOL
- CYBIL
- The EDIT_FILE utility
- FORTRAN
- Object code management
- Screen Formatting
- Source code management
- The System Command Language

Using this manual, you may view, print, or copy an example to a file. Once an example has been placed in a file, it may be executed or edited. To access the Examples manual, enter the following:

```
/help m=examples
```

Once within the manual, you may get instructions on how to read and use the manual by doing the following:

- Press the key(s) that perform the **Help** operation.
- Wait for the help window to appear on your screen.
- Press **Help** again.

NOTE

The examples in the online Examples manual are provided to demonstrate possible uses of NOS/VE and its products. To whatever extent an example seems useful, you are welcome to use it. However, Control Data makes no claims regarding the suitability of these examples for any particular purpose.

Catalog and File Management

4

NOS/VE File and Catalog Structures	4-1
Temporary File Catalogs	4-2
Permanent File Catalogs	4-3
Referencing File Names	4-5
Working Catalogs	4-7
Requirements for Referring to Files	4-8
File Cycles	4-10
File Positioning	4-12
Creating Catalogs and Files	4-13
Creating Catalogs	4-13
Deleting Catalogs	4-15
Creating Files	4-15
Implicit File Creation	4-16
Explicit File Creation	4-17
Deleting Files	4-18
Catalog Entries	4-18
Displaying Catalog Entries	4-19
Changing Catalog Entries	4-20
Displaying Catalog Information	4-20
Catalog and File Permits	4-21
Access Control Entries	4-22
Permit Groups	4-22
Access Modes	4-23
Share Requirements	4-24
Creating and Deleting Access Control Entries	4-26
Multiple Access Control Entries	4-28
Attaching and Detaching Permanent Files	4-30
Attaching Files Explicitly	4-32
Accessing the File	4-32
Sharing the File with Others	4-32
How Sharing of Files Is Controlled	4-33
Multiple Attaches within a Job	4-36
Attaching Files as a Private Reader	4-37
Using the \$ASIS Open Position	4-38
Using Files with Passwords	4-39
NOS/VE File Attributes	4-40
Attributes for All NOS/VE Files	4-40
Establishing File Attributes	4-58

Attributes for Record Access Files	4-59
Attributes for Keyed Files	4-60
Changing File Attributes	4-61
Displaying File Attributes	4-62
Copying Files	4-64
File Organization Combinations	4-66
Copying Sequential Files to Sequential Files	4-67
Copying Sequential Files to Keyed Files	4-71
Copying Byte-Addressable Files to Byte-Addressable Files	4-73
Copying Keyed Files to Keyed Files	4-74
Copying Keyed Files to Sequential Files	4-77
Copying List Files	4-79
Input File Has a FILE_CONTENTS Value of LIST	4-79
Output File Has a FILE_CONTENTS Value of LIST	4-79
Comparing Files	4-80
Displaying the Representation of Data in a File	4-82
Managing Remote Files and Catalogs	4-84
Implicit Remote NOS/VE File Access	4-84
Creating a Remote Validation	4-84
Using Commands for Implicit Remote NOS/VE File Access	4-85
Explicitly Accessing Remote Files	4-86
Transferring a Remote File to Your System	4-88
Transferring a File to a Remote System	4-89
Common Remote File Management Problems	4-90

This chapter describes the NOS/VE file system and the concepts required to use and manage files. The following are described in detail:

- NOS/VE Catalog and File Structures
- Referencing File Names
- Creating Catalogs and Files
- Catalog and File Permits
- Attaching and Detaching Permanent Files
- NOS/VE File Attributes
- Copying Files

The EDIT_CATALOG utility, described in the Introduction to NOS/VE manual, makes use of the concepts described in this chapter; however, that utility is a screen mode application and uses labelled function keys to perform the file management operations that are described in this chapter.

NOS/VE File and Catalog Structures

NOS/VE stores all programs and data in files. Files, in turn, are stored in logical groupings called catalogs.

NOS/VE supports permanent and temporary files. These files reside in permanent and temporary file catalogs.

Temporary File Catalogs

Your temporary catalog, called \$LOCAL, is a temporary workspace for the temporary files you create during your job. In addition, NOS/VE uses the \$LOCAL catalog to create and store certain system-defined files that it requires to process your job.

Certain temporary files in the \$LOCAL catalog are referred to as local files in this manual. These are the files that are given a local file name and are then stored in the \$LOCAL catalog under that file name. Local files are attached to a job by means of a local file name and are then considered "local" to the job that attached them.

The \$LOCAL catalog contains files only; you cannot create catalogs within \$LOCAL.

The \$LOCAL catalog exists only during the time your job exists. All the files in \$LOCAL are deleted when your job ends.

Permanent File Catalogs

Within NOS/VE, permanent files reside in a hierarchy of permanent catalogs. The permanent catalog that contains all your permanent files and any other permanent catalogs you create is called your master catalog; by default, your master catalog is reserved for your user name when you are validated as a NOS/VE user. The official name of your master catalog is your user name prefixed by a period (.). For example, to obtain a display of the contents of his master catalog, user SARETT enters the following:

```
display_catalog catalog=.sarett
```

The following describes the structure of the NOS/VE permanent files and catalogs:

Name	Description
Family	Designates a grouping of NOS/VE users. Each NOS/VE system can have more than one family. You can refer to your family with \$FAMILY.
Master catalog	Contains all the permanent files and catalogs that belong to your user name. You can also refer to this catalog with \$USER.
Catalog	Designates a permanent catalog which is subordinate in the hierarchy to the master catalog.
File	Designates a collection of information referenced by a name; the file resides on its own or within a permanent catalog within the master catalog.

You can picture this hierarchy as shown in the following figure:

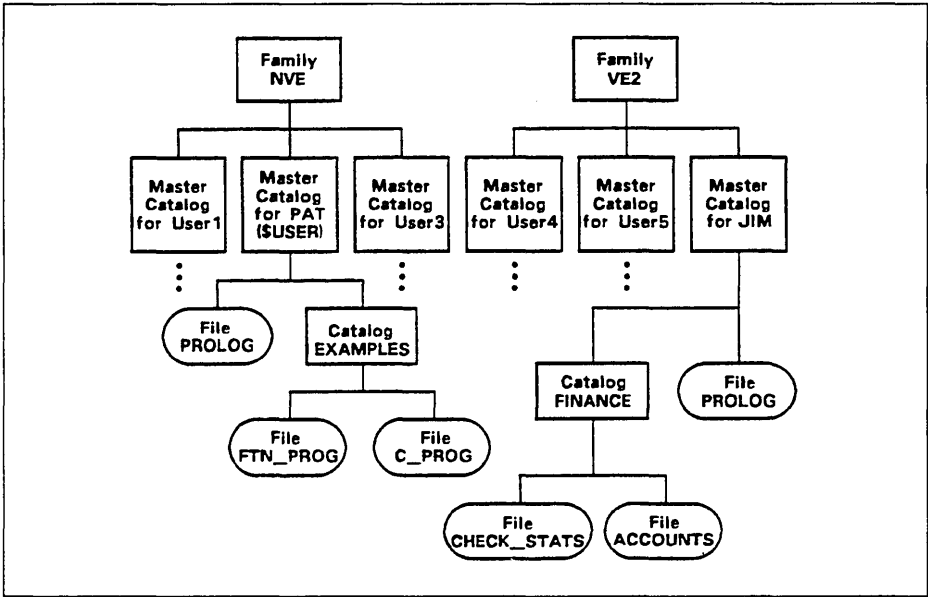


Figure 4-1. Master Catalog Structure.

Referencing File Names

To manipulate and perform operations on NOS/VE files, you often must refer to the file as a file name on a NOS/VE command. You do this by specifying its *file path*. A complete file path can include the family name and all the catalogs in the path that lead to the file. A file path can also specify an optional file cycle as well as an optional open position. Conceptually, you can think of each item in the file path as an element. The most complete file path you can use includes the following elements:

```
:family.username.catalog(s).filename.cycle.openposition
```

It is not usually necessary for you to use the complete file path shown in the preceding example. As stated previously, file cycles and open position elements are optional. File cycles and open positions are described in a later section.

The best way to trace a file path is to start with the file name and trace the path to the top of the file hierarchy. Suppose in the preceding figure that user PAT wants to copy the file FTN_PROG to temporary file SCRATCH. The following example shows the complete file path to filename FTN_PROG:

```
/copy_file input=:nve.pat.examples.ftn_prog output=$local.scratch
```

You can refer to files within your own family or within other users' family catalogs. Suppose you want to copy user JIM's CHECK_STATS file, and user JIM is part of family VE2. The correct way to reference the file, then, is as follows:

```
/copy_file input=:ve2.jim.finance.check_stats
../output=$local.scratch
```

Note that the above command will result in an error if you are not user JIM or JIM has not permitted you access to CHECK_STATS. See Catalog and File Permits later in this chapter for more information.

A file path can also contain a *file cycle*. A file cycle is a numbered version of a given file. That is, whenever more than one copy of a file exists at the same time, by default the copies will be numbered in the order that they were created. Thus, if two versions of file CHECK_STATS existed, you could print version two by specifying the following file path:

```
/print_file file=:ve2.jim.finance.check_stats.2
```

You can also specify an optional *open position* element. For example, suppose you are user JIM and have some statistics in file TEMP that you want to copy to the end of file CHECK_STATS. The complete file reference is:

```
/copy_file input=$local.temp output=..  
./:ve2.jim.finance.check_stats.$eo
```

NOS/VE provides a number of ways to refer to catalogs and files without specifying a complete file path. The following sections describe the short cuts, provide in detail the basic rules for referring to NOS/VE files and catalogs, and describe in further detail the concepts of file cycles and file positioning.

Working Catalogs

Your working catalog is the catalog in which NOS/VE assumes a file exists if you do not specify its complete file path. When you log in, NOS/VE normally sets your working catalog to \$LOCAL. However, this can be changed through the use of either your own user prolog, or the prologs maintained by your site. For more information on prologs, see Job Management later in this manual.

Setting your working catalog to a catalog within your master catalog provides two benefits:

- You can work within your master catalog without having to enter the family and catalog elements of the file path.
- Any files and catalogs you create are automatically saved when you log out because you are working within your own master catalog.

To set your working catalog to a specific catalog, specify the catalog's path on the SET_WORKING_CATALOG command. For example, user PAT sets the working catalog to EXAMPLES by entering the following:

```
/set_working_catalog catalog=.pat.examples
```

or uses the equivalent, which is:

```
/set_working_catalog catalog=$user.examples
```

Now user PAT can create files by entering far fewer keystrokes as in the following:

```
/create_file file=pascal_program
```

The PASCAL_PROGRAM file is automatically saved in user PAT's EXAMPLES catalog.

You can change your working catalog to any other catalog as often as you wish during your job. You can return the path of your current working catalog by using the \$CATALOG function.

Requirements for Referring to Files

While it is never incorrect to specify a full catalog or file path, it can often be inconvenient. The following list describes the short cuts that NOS/VE accepts for referring to a file path:

- Specify the name of the family in the file path only if you are referencing a catalog or a file residing under a family different than yours.
- Specify the name of a master catalog in the path only if you are referencing a catalog or file that does not reside within your working catalog, or within a catalog subordinate to your working catalog.
- Use the \$LOCAL keyword only if your \$LOCAL catalog is not your working catalog.
- Specify just the name of the catalog or file if the catalog or file resides in your working catalog.

Note that the file path you specify determines what characters you use to begin the file reference. The following table describes the possible beginnings for a file path.

Value	Meaning
Colon (:)	Used when the family catalog is included in the file reference.
Period (.)	Used when the master catalog begins the file reference.
\$FAMILY	Used to begin a reference to a file or catalog in your family catalog.
\$USER	Used to begin a reference to a file or catalog in your master catalog.
\$LOCAL	Used to begin a reference to your local catalog.
\$SYSTEM	Used to reference :\$SYSTEM.\$SYSTEM (that is, family \$SYSTEM, user \$SYSTEM). This username contains files used to support many NOS/VE products.
File or Catalog name	Path begins with the name of a catalog or file.

For example, consider the following figure:

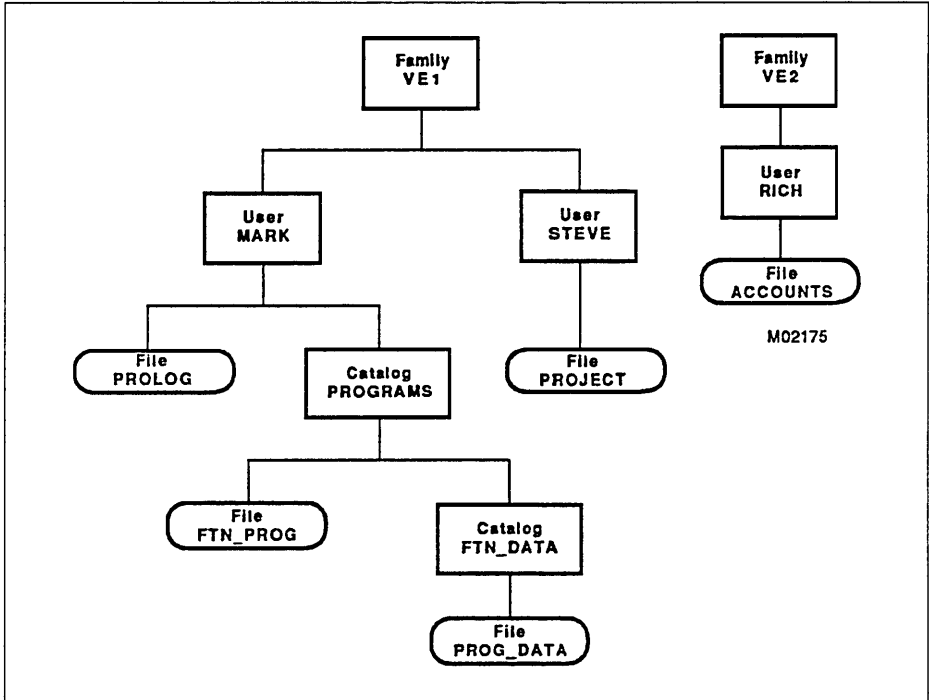


Figure 4-2. Two Family File Hierarchy

Suppose that user MARK sets his working catalog to catalog PROGRAMS by entering:

```
/set_working_catalog catalog=.mark.programs
```

Mark can then access the following files with the following references:

**Path from
PROGRAMS**

Full Path Reference

\$user.prolog	:ve1.mark.prolog
ftn_prog	:ve1.mark.programs.ftn_prog
ftn_data.prog_data	:ve1.mark.programs.ftn_data.prog_data
.steve.project	:ve1.steve.project
:ve2.rich.accounts	:ve2.rich.accounts

File Cycles

A permanent file can have up to 999 different versions. Each version, called a cycle, is identified by a cycle number. When you create a file, NOS/VE identifies the version of the file as cycle 1 unless the file previously had a cycle created for it, or you explicitly reference a cycle number. If the file previously had a cycle created for it, NOS/VE will create the new file with the next highest available cycle number.

You can make a reference to a specific file cycle by appending the cycle reference to the end of the file reference. The reference can be in the form of an unsigned integer, or a keyword that represents a specific cycle number. The functions that may be used are as follows:

Cycle Descriptor	Meaning
\$HIGH	The highest currently defined cycle number. This designator can only be used for existing files.
\$LOW	The lowest currently defined cycle number. This designator can only be used for existing files.
\$NEXT	A new cycle of a file with a cycle number one greater than that designated by \$HIGH. If the file doesn't exist, 1 is used.
unsigned integer	The cycle whose number is specified. The number must be in the range of 1 through 999.

For example, suppose you have cycles 1 and 2 of file STEVE_PGM. The following both refer to cycle 2 of STEVE_PGM:

```
steve_pgm.2
```

```
steve_pgm.$high
```

In the same manner, the following entries both refer to cycle 1 of STEVE_PGM:

```
steve_pgm.1
```

```
steve_pgm.$low
```

If the specified cycle of a file referenced for an output operation does not exist, NOS/VE creates the requested cycle.

To create another cycle of STEVE_PGM you can use the COPY_FILE command as follows:

```
/copy_file input=steve_pgm output=steve_pgm.$next
```

As a result of the copy, the version of the file that is one number higher than the current highest version of STEVE_PGM is created.

Normally, if you omit the cycle reference on the file path, NOS/VE uses \$HIGH. However, always check the default file cycle reference you specify on a file path. For example, if you omit the file cycle reference on the FILE parameter of the DELETE_FILE command, NOS/VE deletes the cycle which corresponds to \$LOW.

You can change a file's cycle number by using the CHANGE_CATALOG_ENTRY command. For example, to change cycle 2 of file STEVE_PGM to cycle 88, enter the following:

```
/change_catalog_entry file=steve_pgm.2 new_cycle=88
```

File Positioning

The position of a file is the location in the file at which data is being written and read. A file may be positioned at its beginning, end, or somewhere in between. You can include a file position on a file reference to indicate how you want the file positioned when it is opened. This open position is always the last element of a file reference. The possible open positions are listed as follows:

Open Position	Meaning
\$BOI	Positions the file to the beginning-of-information when it is opened.
\$EOI	Positions the file to the end-of-information when it is opened.
\$ASIS	Meaningful only on files that are explicitly attached using the ATTACH_FILE command. See Attaching and Detaching files later in this chapter.

NOTE

The open position specified on the file reference overrides any open positioning specified on a previous command.

Creating Catalogs and Files

By using the appropriate NOS/VE commands, you can create and delete files and catalogs residing in your master catalog. You can also create files for the temporary catalog, \$LOCAL. Because you are the owner of your master catalog as well as the files and catalogs within your master catalog, you can permit other users different kinds of access to your permanent files and catalogs. File and catalog permits are described later in this chapter.

NOS/VE uses the following rules to determine ownership of a catalog or file:

- You are the owner of files or catalogs within your master catalog.
- You are the owner of files and catalogs registered in subordinate catalogs within your master catalog.

You cannot create permits allowing others to use the files in \$LOCAL.

Creating Catalogs

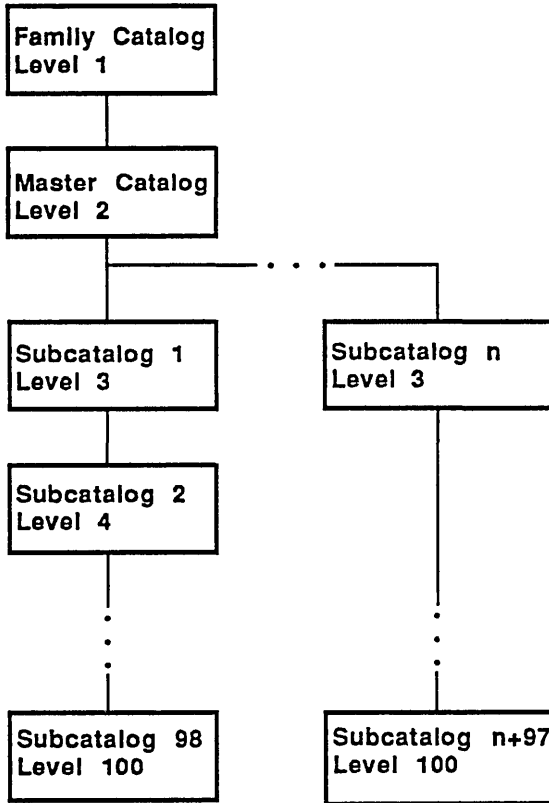
You can create new catalogs within your master catalog or within catalogs subordinate to your master catalog using the CREATE_CATALOG command. This command allows you to create catalogs explicitly; NOS/VE does not support implicit catalog creation.

For example, user MARK can create catalog NEW_CATALOG by entering the appropriate path on the CREATE_CATALOG command. The last element of the path is the name of the created catalog.

```
/create_catalog c=.mark.programs.new_catalog
```

As a result of the preceding example, catalog NEW_CATALOG is created and registered in catalog PROGRAMS.

A NOS/VE user may create a path that is 100 catalogs deep. However, the family and master catalogs each count for a level. For example, consider the following figure:



M02177

In addition, the actual path name can have no more than 256 characters in it. Thus, a path can be 100 catalogs or 256 characters long, whichever comes first.

The total number of catalogs that you may create within your master catalog is limited by the amount of mass storage memory your site allows you to use.

Deleting Catalogs

You can delete a catalog residing within your master catalog using the `DELETE_CATALOG` command. For instance, in order to delete catalog `PROGRAMS` (and everything in it), user `MARK` would enter the following:

```
/delete_catalog catalog=.mark.programs ..  
../delete_option=catalog_and_contents
```

Once the catalog is deleted, `NOS/VE` releases all storage space associated with the catalog and deletes the catalog name from within the master catalog.

In order to delete a catalog, you must be the owner of it. In addition, you cannot delete your master catalog.

The `DELETE_CATALOG` command may also be used to delete all of a catalog's contents while leaving the catalog itself untouched. In order for Mark to delete catalog `PROGRAMS`' contents and leave `PROGRAMS` itself untouched (that is, make `PROGRAMS` an empty catalog), he would enter:

```
/delete_catalog catalog=.mark.programs ..  
../delete_option=contents_only
```

Creating Files

As is the case with catalogs, you can create files within a catalog for which you are the owner. `NOS/VE` recognizes two methods of file creation: explicit and implicit.

Implicit File Creation

Implicit file creation may be accomplished with the use of the following NOS/VE commands:

- COPY_FILE
- EDIT_FILE
- COLLECT_TEXT

Each of these commands contains a parameter that allows you to supply a non-existing file name; NOS/VE uses that file name to create a new file. In the following example, the file named on the OUTPUT parameter is created as a result of entering:

```
/copy_file input=.jim.finance.check_stats ..  
../output=.jim.finance.new_stats
```

The following command creates the file named on the FILE parameter. You can then use the EDIT_FILE utility to enter text in the newly created file.

```
/edit_file file=.jim.finance.new_stats
```

You can also use the EDIT_FILE command to edit an existing file and then create a different file containing the edited version of the existing file. You do this by first using the EDIT_FILE utility to edit the file. Then, once you are done editing the file, use the WRITE_FILE subcommand to write the contents of the file to a new file. See the NOS/VE File Editor manual for more information on the EDIT_FILE and WRITE_FILE commands.

If you use the COLLECT_TEXT command to create a file, NOS/VE prompts you for line-by-line text insertion. The following shows the implicit creation of file TEXTFILE, and the use of the ct? prompt. To stop entering text, enter two asterisks (**). For example:

```
/collect_text o=$user.textfile  
ct? This is some information that I want to place  
ct? into file TEXTFILE in my master catalog.  
ct? **  
/
```

In addition to the commands listed here, you may implicitly create a file using any command that writes information to a file. Thus, the `OUTPUT` parameter that exists on all of NOS/VE's `DISPLAY` commands can be used to create a file. For instance, user `MARK` can create a file containing a list of every catalog and file immediately subsequent to his master catalog by entering the following:

```
/display_catalog catalog=.mark output=.mark.disc_output
```

Explicit File Creation

You can create files explicitly using the `CREATE_FILE` command. `CREATE_FILE` provides parameters with which you can specify a permanent file name, a local file name, a password for the file, and a retention period for a file cycle; you can also specify that the system should keep a log of file access activity.

In addition to creating a file, the `CREATE_FILE` command explicitly attaches the new file to your job. That is, after the command is executed, your job can access the file through the local catalog.

You may detach the file from your job explicitly using the `DETACH_FILE` command. For information about attaching and detaching files, refer to *Attaching and Detaching Files* later in this chapter.

In general, you should use the `CREATE_FILE` command only when you need to create a new file under one or more of these conditions:

- You want to specify a password for a new file or create a new cycle for a file that has a password.
- You want to specify the length of time a new file cycle is retained by the system.
- You need to assign a local file name to the file for use by subsequent commands in a program.
- You want to specify the `$ASIS` file position when you reference the file on subsequent commands in the job.

Deleting Files

As is the case with catalogs, you can delete files for which you are the owner. In addition, it is also possible for you to delete files you don't own if you are permitted to do so by the owner of the file. See *Catalog and File Permits* later in this chapter for more information.

When you delete a file, the system releases all storage space for the file. If you attempt to delete a file that is being used by another job, the file remains attached to that job until it is detached; however, the job issuing the `DELETE_FILE` command can no longer access the file. If the file is attached within a job but is not opened, a `DELETE_FILE` command will also detach the file from the job.

NOTE

When deleting a file, be certain to query the file cycles in existence for that file and indicate what cycle you want to delete. If you do not specify the cycle number, `NOS/VE` deletes the lowest file cycle.

Catalog Entries

In `NOS/VE` when a permanent file is created it is registered in a catalog, and certain kinds of information are recorded and preserved with the file name. `NOS/VE` maintains this information as a *catalog entry* for the file. Catalog entries include the following:

- File name
- Optional file password
- Access control list
- Optional access log
- Account and project information
- Cycle descriptor for each cycle of the file

The access log is a record of all the users who access the file. This log contains the following information for each user who accessed the file:

- The user name of the user who accessed the file.
- The number of times the user accessed the file.
- The last cycle accessed by the user.
- The date and time of the last access by the user.

The cycle descriptor provides the following information:

- The file's cycle number.
- The number of times the file has been accessed.
- The date and time the file was created, last accessed, and last modified.
- The expiration date for the file.
- File attributes

Access control lists are described in Catalog and File Permits later in this chapter. File attributes are described in the File Attributes section later in this chapter.

Displaying Catalog Entries

You can display the information contained in a file's catalog entry by using the `DISPLAY_CATALOG_ENTRY` command. If the file belongs to another user, you can display the information in the catalog entry only if you are permitted access to the file.

If you display the file's cycle description, you will see dates and times for the last access and modification of the file. Note, however, that if you currently have explicitly attached the file (explicit file attachments are described later in this chapter), the date and times fields will show the date and time of the attachment.

For example, if you explicitly attach a file and then edit it, the last access and modification times will show the date and time of the attachment, not the edit. When you explicitly detach the file, the last access and modification fields will be updated to show the date and time of the detachment.

You can use the `DISPLAY_FILE_ATTRIBUTES` command to display a file's attributes.

Changing Catalog Entries

You can use the `CHANGE_CATALOG_ENTRY` command to change the following catalog entries:

- The file name, password, log selection, and account and project identification for all cycles of a file.
- The retention period, cycle number, and damage condition for a specific cycle of a file.

You can change a file's attributes using the `CHANGE_FILE_ATTRIBUTES` commands.

You can change a catalog entry for a file only if you have control access for the file. If the file has a password associated with it, you must also specify the password.

Displaying Catalog Information

You can display information about a catalog by using the `DISPLAY_CATALOG` command. `DISPLAY_CATALOG` displays the following information:

- The names of all catalogs and files registered in the catalog.
- A description of the files registered in the catalog including information about the files' cycles, or account and project information.
- A description of the access control entries established for the catalog. Access control entries are described later in this chapter.
- A list containing the size in bytes of each file or cycle in the catalog as well as the damage condition of each damaged cycle.

Catalog and File Permits

Through the use of catalog and file permits, you can specify the ways in which you are willing to share your files with other NOS/VE users. You can also limit access to catalogs and files for which you are the owner, both to yourself and to others. When you create a file permit, the permissions you define for the file are recorded by NOS/VE as an access control entry for the file. The access control entries defined for a file constitute a file's *access control list*.

You cannot create permits allowing others to use the files in \$LOCAL.

An access control entry defined for a catalog applies to each file in the catalog as well as to each file in any subordinate catalogs registered in the catalog. However, even if a file resides in a catalog for which an access control entry is already defined, you may define a separate set of permits for the file for a particular user or a particular group of users.

Only the owner of a file or a catalog may establish a permit for it. Although a catalog permit can define the access control entry for the files in the catalog, access to the catalog is not controlled by the access control entry.

This section describes the elements of an access control entry, the ways you define access control entries, and the rules for accessing files and catalogs which have file and catalog permits.

Access Control Entries

NOS/VE uses the access control entry defined for a file to determine whether a user or group of users may use the file and the ways in which they may use it and share it with other users. An access control entry contains the following items:

- The user or group of users to which the access control entry applies. In this discussion, the permitted group of users is called the permit group.
- The access modes allowed for the user(s). These access modes specify the ways the file can be used.
- The share modes that must at a minimum be permitted to others by members of the permit group who request access to the file. These modes are called the share modes or share requirements and specify the ways that users must share the file while it is in use.
- Application information that may be used by programs and utilities in whatever ways are necessary. The type and meaning of information appearing in this field is entirely dependent upon the program or utility used to access the file.

Permit Groups

A permit group is the group of users to which an access control entry applies. The following are the permit groups to which an access control entry can apply:

- All users.
- All users in a family.
- One user in a family.
- All users executing under an account name.
- One user executing under an account name.
- All users executing under an account and project name.
- One user executing under an account and project name.

Access Modes

Access modes specify the ways a user may use a file or the files within a catalog. The following are the access modes:

ALL

Access is permitted for APPEND, EXECUTE, MODIFY, READ, SHORTEN, and WRITE.

APPEND

Access is permitted to append information to the end of the file or files.

CONTROL

Access is permitted to delete a cycle and to change its identity including the file name, cycle number, password, log selection, retention, account name, project name, and file attributes.

CYCLE

Access is permitted to add new cycles to the file or files or to create a new file.

EXECUTE

Access is permitted to execute object code or an SCL procedure in the file or files.

MODIFY

Access is permitted to alter data within the existing file or files.

NONE

Access is specifically prohibited.

READ

Access is permitted to read the file or files.

SHORTEN

Access is permitted to delete information from the end of the file or files.

WRITE

Access is permitted for APPEND, MODIFY, and SHORTEN modes.

You may also specify a permission that is a combination of the above access modes. For instance, the permission of READ and EXECUTE is commonly used.

Share Requirements

As the owner of a file, you may want to require that other users of your file make the file available for certain modes of access. To do this you must specify what share modes users must grant to others if the file is used concurrently. This set of share modes established in the file's access control entry constitutes a file's share requirements.

For example, suppose you create an access control entry and require that the file be shared for READ and EXECUTE access. If a user then attempts to access the file and specifies only a READ share mode, the attempt will fail. If you determine that your file does not require sharing by concurrent users, you should specify a share requirement of NONE in the access control entry.

The following are the share modes that can be specified as a file's share requirements:

ALL

Sharing is required for APPEND, EXECUTE, MODIFY, READ, SHORTEN, and WRITE.

APPEND

Sharing is required for APPEND access.

EXECUTE

Sharing is required for EXECUTE access.

MODIFY

Sharing is required for MODIFY access.

NONE

A user is authorized exclusive access, and no modes of sharing are imposed on users of the file. When attaching the file, users may select a share mode of NONE.

READ

Sharing is required for READ access.

SHORTEN

Sharing is required for SHORTEN access.

WRITE

Sharing is required for APPEND, MODIFY, and SHORTEN modes.

Creating and Deleting Access Control Entries

NOS/VE creates a default access control entry for each user's master catalog. A user can delete this default catalog permit or change it to a more restrictive permit allowing access, for example, to just read and execute the files in the catalog.

You use the following commands to create, change, delete, and display access control entries:

CREATE_CATALOG_PERMIT

Creates an access control entry for a catalog. This permit applies to all files and catalogs within the catalog that do not have permits specifically defined for them. This command is also used to change a catalog permit.

CREATE_FILE_PERMIT

Creates or changes an access control entry for a file.

DELETE_CATALOG_PERMIT

Deletes an access control entry for a catalog.

DELETE_FILE_PERMIT

Deletes an access control entry for a file.

DISPLAY_CATALOG

Displays, among other things, the access control list entries for a catalog.

DISPLAY_CATALOG_ENTRY

Displays, among other things, the access control list entries for a file.

You may create, change, and delete access control entries for catalogs and files that you own; these are the files and catalogs that reside in your master catalog. You can permit groups of users access to your files, and you can also create more restrictive kinds of access to specific users.

For example, suppose you are user SARETT and you want to permit user SUEO to write to file CHECK_STATS which resides in catalog EXAMPLES within your master catalog. Let us say that user SUEO is part of a group which already has READ and EXECUTE access to all the files in catalog EXAMPLES. The permissions for all the files in catalog EXAMPLES are displayed by entering:

```
/display_catalog catalog=.sarett.examples display_option=permits
```

```
PERMIT_GROUP: PUBLIC
PERMITS: READ, EXECUTE
SHARE: NONE
APPLICATION_INFORMATION:
```

At this point, the files in catalog EXAMPLES are for public use and may be read and executed. User SARETT can then enter the following to permit user SUEO to write to file CHECK_STATS:

```
/create_file_permit file=.sarett.examples.check_stats ..
../group=user user=sueo access_mode=write share_mode=none
```

Note, however, that SUEO will no longer be able to read or execute the file.

When you are creating access control entries, you can also restrict your own access. You can do this by specifying your user name for the USER parameter on the CREATE_FILE_PERMIT command. For example, to restrict your own access to file CHECK_STATS in catalog EXAMPLES, enter the following:

```
/create_file_permit file=.sarett.examples.check_stats
../group=user user=sarett access_mode=read share_mode=read
```

NOTE

If both the group and the user parameters are not specified on the CREATE_FILE_PERMIT command, a permit will be created using your user name as a default.

No matter how much you restrict your access to a catalog or file that you own, as the owner, you can always delete the catalog's or file's access control entries.

When you are creating file and catalog permits, keep in mind that you can access a given file with a NOS/VE command only if the minimum access permissions for the file conform to the operation of the command. For example, the DELETE_FILE command requires that the user have CONTROL permission. In addition, the following are also true:

- Commands that write files require NONE as the share requirement in effect for the file.
- Commands that read files require the absence of APPEND, MODIFY, SHORTEN, and WRITE from the share requirements in effect for the file.

Multiple Access Control Entries

A catalog or file can have many access control entries defined for it. In addition, a catalog or file may have access control entries defined for each element in its path. When this situation occurs, the system uses the following rules to determine which access control entry to apply to the user:

- If you belong to more than one permit group for which an access control entry is defined, the access control entry applicable to the smaller group applies.

For example, if one entry applies to all users (that is, the permit group is PUBLIC) and another permit group applies to a family of users to which you belong, the system uses the entry applying to the family of users.

- If access control entries for the same permit group exist for more than one element of the catalog or file path, the entry applicable to the last element in the path applies.

For example, consider the following path:

```
$user.catalog_1.catalog_2.junk_file
```

Suppose the master catalog and CATALOG_1 each have a single access control entry and they both apply to the same permit group, and neither CATALOG_2 nor JUNK_FILE have access control entries. In that case, the permissions defined in CATALOG_1's access control entry apply to both JUNK_FILE and all the files in CATALOG_2.

If necessary, the system uses both rules to determine the applicable access control entry. For example, consider the following file path:

```
catalog_1.file_1
```

Suppose CATALOG_1 has an access control entry applying to FAMILY_A. Further, suppose that FILE_1 has access control entries applying to the PUBLIC permit group (which is all users) as well as to FAMILY_A. If a member of the permit group, FAMILY_A, attempts to access FILE_1, the following factors are considered:

1. CATALOG_1's access control entry can not be used because it is not the last applicable entry in the file reference.
2. FILE_1's access control entry defined for the PUBLIC permit group does not apply because it does not involve the smallest permit group.
3. This leaves FILE_1's access control entry for FAMILY_A which is the access control entry the system uses to grant access to the users in FAMILY_A for FILE_1.

Attaching and Detaching Permanent Files

When a permanent file is needed for use within a job, the job must attach it. When the file is attached, it becomes reserved for use within the job. When the job is finished using the file, it should be detached.

Attachment is achieved either explicitly or implicitly. When you make a reference to a file on a NOS/VE command, NOS/VE automatically attaches the file to the job. When the command operation is complete, NOS/VE automatically detaches the file from the job. These file attachments and detachments are somewhat transparent to the user. This method of attaching and detaching files is called implicit attachment.

In NOS/VE, explicit file attachment is recommended for files that require sharing within a job. You may also want to explicitly attach files for the following reasons:

- You need to specify the file's password.
- You need to specify a local file name for the file to be used by subsequent commands in the job.
- You need to use the \$ASIS file position on subsequent references to the file.
- You need to restrict certain kinds of access to the file while your job is using it.

The following NOS/VE commands are available to attach and detach your permanent files explicitly:

ATTACH_FILE

Use this command to attach a permanent file for which you specify an optional local file name, a password, the modes of access in which you intend to use the file, the modes in which you are willing to share the file, and whether you are willing to wait if the file is currently unavailable to you.

CREATE_FILE

Use this command to create a new file and attach it to a job with an optional local file name, password, and enable NOS/VE to log the activity of the new file.

DETACH_FILE

Use this command to detach a file from a job.

The remainder of this section describes the rules NOS/VE uses to determine whether file access is granted to attach requests.

Attaching Files Explicitly

When you use the `ATTACH_FILE` command to attach a file, you may specify the ways you want to use the file. You may also specify the ways in which you are willing to share the file within your job and within other jobs requesting use of the file. You use the `ACCESS_MODES` and `SHARE_MODES` parameters to do this.

Accessing the File

When you explicitly attach a file you use the `ACCESS_MODE` parameter to specify how you intend to use the file while it is attached to your job. For example, if you want to use the file for the `READ` and `MODIFY` access modes, you then enter the following:

```
/attach_file file=.sarett.examples.file_a ..  
../access_mode=(read,modify)
```

`NOS/VE` then compares the above request with the set of permissions established by the file's owner in the file's access control entry. If `READ`, `MODIFY` access is not among the access permissions in the file's access control entry, the attach attempt will fail. Note that the default value for the `ACCESS_MODE` parameter is `READ` and `EXECUTE`, and that you may specify neither `CYCLE` nor `CONTROL` access for this parameter.

Sharing the File with Others

The share requirements specified on the file's access control entry are the minimum ways the file's owner has specified that the file must be shared. Thus, you must include these share requirements in the set of share modes specified on the `SHARE_MODES` parameter of the `ATTACH_FILE` command. You can always specify a share mode set that is larger than the set specified on the file's access control entry.

In the preceding example, the requested attachment of FILE_A is successful only if the FILE_A's owner specified NONE for the share requirements in the file's access control entry. For example, let's say the file's owner established that FILE_A must, at a minimum, be shared for READ access. Then the preceding example must be expanded to include the SHARE_MODES parameter:

```
/attach_file file=.sarett.examples.file_a
../access_mode=(read,modify) share_mode=read
```

Note that the default values supplied by the ATTACH_FILE command for the SHARE_MODE parameter depend upon the value specified on the ACCESS_MODE parameter. If the ACCESS_MODE parameter includes APPEND, SHORTEN, or MODIFY, the SHARE_MODE parameter will default to NONE. Otherwise, the SHARE_MODE parameter defaults to READ and EXECUTE.

How Sharing of Files Is Controlled

When a job attempts to attach a file that is already attached to job, the following conditions must be met in order for the attachment to succeed:

- The maximum access modes that you can specify on the attach request is the intersection of the following sets:
 - The set of access modes established in the applicable access control entry.
 - The intersection of all share mode sets specified on outstanding attachments of the file.
- The minimum number of share modes you must specify on the attach request is the union of the following sets:
 - The set of share requirements established in the applicable access control entry.
 - The union of access mode sets specified on all outstanding attachments of the file.

For instance, suppose FILE_A has the following access control list:

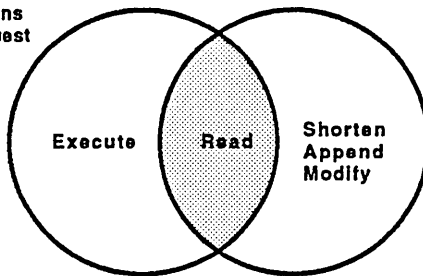
```
/display_catalog_entry ..  
../file=.sarett.examples.file_a display_option=permit  
PERMIT_GROUP: USER  
  FAMILY: NVE, USER: RICH  
  PERMITS: READ, SHORTEN, APPEND, MODIFY, EXECUTE  
  SHARE: NONE  
  APPLICATION_INFORMATION:  
PERMIT_GROUP: USER  
  FAMILY: NVE, USER: PAT  
  PERMITS: READ, SHORTEN, APPEND, MODIFY  
  SHARE: READ, EXECUTE  
  APPLICATION_INFORMATION:
```

If users RICH or PAT want to attach FILE_A, they must request the attachment of the file with the access modes as well as the share requirements established for their applicable access control entry. For example, if FILE_A is not currently attached, user RICH can specify the following and be granted an attachment of FILE_A:

```
/attach_file file=file_a local_file_name=first_a ..  
../access_modes=(read,append,shorten) share_modes=(read,execute)
```

Now, suppose user PAT wishes to attach FILE_A while user RICH currently has it attached with the previous selections. Then her allowable access and minimum share requirements are determined by the rules stated previously in this section. For instance, PAT's allowed access to the file as determined by these rules is represented by the shaded area in the following figure:

**Rich's Share Selections
on the ATTACH Request**

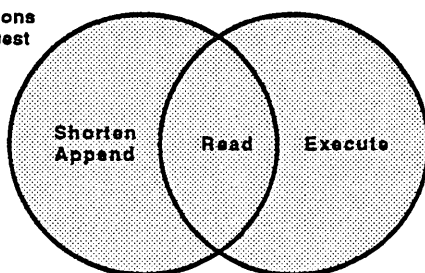


 Represents Pat's access to the files as allowed by NOS/VE.


MO2246

Further, PAT's minimum share requirements are represented by the shaded area in the following figure:

**Rich's Access Selections
on the ATTACH Request**



**Pat's Share Requirements
in the File Permit**

 Represents the minimum share mode required for Pat by NOS/VE.

M02247

Thus, the following attach request represents the only allowable access and minimum share selections that PAT can make for FILE_A in this situation:

```
/attach_file f=file_a lfn=file_1 am=read ..
../sm=(read,shorten,append,execute)
```

The WAIT parameter on the ATTACH_FILE command offers the choice to wait for a file which may be currently unavailable for certain modes of access or sharing due to a current attachment of the file.

Multiple Attaches within a Job

You can attach a file more than once within a job. To do this you must specify a unique local file name on the LOCAL_FILE_NAME parameter of the ATTACH_FILE command for each attachment within the job. If the name is already known within the job, the attach will fail. Also, for best results you should avoid names that are the same as an SCL variable defined within the job.

A file may be attached to a job more than once either by a single task or by a group of tasks. For the purposes of this discussion, assume that several tasks are sharing access of an attached file. Keep in mind that the following discussion also pertains to a single task which has multiple attachments of a file.

Once a file is attached by two or more tasks within a job, what happens most often is that the two tasks share the same file position; that is, the tasks read and write data from the same position in the file. If no effort is made to coordinate the positioning of the file among multiple tasks, a second task's requested file position can also affect the file positioning of tasks which already have the file attached.

For example, suppose TASKA is writing information to FILE_A, and TASKB attaches the file with the default open position, \$BOI (beginning-of-information). TASKA's file position will also be positioned at FILE_A's beginning-of-information. For this reason, care must be taken to coordinate the activities of each task concurrently accessing a file.

There are two ways to avoid this situation. The first is for TASKB to attach the file with an OPEN_POSITION of \$ASIS. This way, TASKB's file position becomes the same as TASKA's. The second way is for TASKB to attach the file as a private reader. See Using the \$ASIS Open Position later in this section for more information on the \$ASIS open position. Private readers are discussed next.

Attaching Files as a Private Reader

A private reader is a task that does not want to share the file position of other tasks currently accessing the file. To be a private reader, all of the following must occur:

- A task must attach the file with the `ATTACH_FILE` command.
- The file may only be attached for `READ` and/or `EXECUTE` access by the task that desires to privately read the file.
- The file's organization must be either sequential or byte-addressable.
- A value of `TRUE` must be specified on the `PRIVATE_READ` parameter of the `ATTACH_FILE` command. This value may be specified explicitly by the task executing the `ATTACH_FILE` command. However, if no value is specified for this parameter, and all of the previous conditions are met, by default `NOS/VE` grants a private read status for the file attachment.

Since private readers of files are neither affected by nor may affect the file positions of other tasks, a private reader may want to exclude `SHORTEN` from its share mode privileges. Otherwise, it is possible for another task or job to shorten the file and cause the private reader to be reading from beyond the end-of-information.

Using the \$ASIS Open Position

If you have explicitly attached a file, you can specify an open position of \$ASIS when you access the file. The \$ASIS position has several different meanings depending upon the circumstances surrounding the access to the file, as described below:

- If the accessing task is a private reader and
 - the file has not been opened within the job before by the accessing task, this open position denotes beginning-of-information.
 - the file has been opened within the job before by the accessing task, this open position denotes the position of the file when it was last closed by the accessing task.
- If the accessing task is not a private reader and
 - the file has not been opened within the job before by any non-private reading task, this open position denotes beginning-of-information.
 - the file is currently opened by another non-private reading task within the job, this file position denotes the current position within the file of the non-private reading task.

For instance, suppose TASKA is a non-private reading task that has opened FILE_1. If TASKB is a non-private reading task that opens FILE_1 with an open position of \$ASIS, TASKB will be positioned to TASKA's current position within FILE_1.

- the file is not currently opened within the job by any non-private reading task, but it has been opened before by a non-private reading task within the job, this open position denotes the file's position when it was last closed by a non-private reading task within the job.

For instance, suppose TASKA has accessed FILE_1 as a non-private reader but has subsequently closed the file. If no other non-private reading tasks within the job are currently accessing FILE_1, and TASKB is a non-private reader that opens FILE_1 with an open position of \$ASIS, the resulting open position will be the same as TASKA's position in FILE_1 immediately before closing the file.

Using Files with Passwords

You can create more security for a file by specifying a password for it. Once this is done, the file can only be accessed by permitted users who specify the file's password. For example, the following creates a password for a `FILE_A`:

```
/create_file file=file_a password=turkey
```

To create a password for an existing file, or to change an existing file's password, use the `CHANGE_CATALOG_ENTRY` command. If you are the file's owner, you may display the file's password using the `DISPLAY_CATALOG_ENTRY` command.

Once a password is created, you must use it on some `NOS/VE` commands. For example if a password was created for a file, the `DELETE_FILE` command will not operate on that file unless the file's password is specified.

For the most part, in order to use a file that has a password, you must explicitly attach the file. When you explicitly attach the file, you must specify its password, and the access and share modes for which you want to use the file. For instance, suppose `FILE_A` has a password of `TURKEY` and you want to edit the file. Then, at a minimum, you must do the following:

```
/attach_file file=file_a password=turkey ..  
../access_mode=(read,write) share_mode=none  
/edit_file file=file_a
```


NOS/VE File Attributes

Each NOS/VE file cycle has a set of file attributes that describe its content and control its processing. File attributes are preserved for the life of a file. Once defined, they need not be reestablished.

There are two types of attributes: preserved and temporary.

- *Preserved attributes* can be classified as follows:
 - Those that cannot be changed during the life of the file cycle. These attributes describe the organization of the information in the file cycle.
 - Those that can be changed permanently. These attributes may describe the content of the file cycle or affect the manner in which the cycle is accessed by a command.
- *Temporary attributes* are those that describe a particular use (attachment) of a file.

The `SET_FILE_ATTRIBUTE` command is used to establish the initial file attributes for a new file. The term new file refers to a file cycle that has never been used (has not been opened for processing). The file cycle is not actually created until it is opened or a `CREATE_FILE` command is executed.

The `ATTACH_FILE` command is used to specify temporary attributes which remain in effect until the file cycle is detached or the job ends.

The `CHANGE_FILE_ATTRIBUTE` command is used to change certain file attributes permanently for a file cycle that has been used (opened for processing).

The `DISPLAY_FILE_ATTRIBUTE` command can be used to display the current attributes of a file cycle.

Attributes for All NOS/VE Files

The following list gives a description of each file attribute:

- `ACCESS_MODES`

Specifies how the file is to be used by subsequent commands that do not explicitly specify an access mode when the file is opened. The following options are available.

READ

You can read the file.

WRITE

You can write the file (combination of APPEND, MODIFY, and SHORTEN).

APPEND

You can append information to the end of the file.

MODIFY

You can alter data within the existing file.

SHORTEN

You can delete data from the end of the file.

EXECUTE

You can execute the file.

NONE

No access to the file is permitted until a subsequent SET_FILE_ATTRIBUTE command restores file access to one or more of the preceding selections.

If the file is currently explicitly attached, this access mode must be a subset of the access mode selections specified with the ATTACH_FILE command.

Omission for a new temporary file causes READ and WRITE to be used.

Omission for an old file causes READ and/or WRITE to be used depending upon whether the ring of the command accessing the file is within the READ and/or WRITE bracket of the file. Rings are discussed in the Object Code Management manual.

Omission for a permanent file that has been scheduled for job access using the ATTACH_FILE command causes the ACCESS_MODE specified on that command to be used as qualified by the ring of the command accessing the file.

- **AVERAGE_RECORD_LENGTH**

Specifies an estimate length of the average record in a new keyed file. This attribute is ignored for a sequential or byte-addressable file and for an old indexed sequential file.

For details, see the SCL Advanced File Management manual.

- **BLOCK_TYPE**

Specifies the block type. This attribute applies only to a sequential or byte-addressable file. Options are:

SYSTEM_SPECIFIED (SS)

The file is logically divided into a number of fixed-sized blocks whose length is determined by NOS/VE. The disk block size is 2,048 bytes. The tape block size is 4,128 bytes. The **MAXIMUM_BLOCK_LENGTH** and **MINIMUM_BLOCK_LENGTH** attributes do not affect this blocking algorithm.

USER_SPECIFIED (US)

The file is logically divided into a number of blocks whose length may vary between a user-defined minimum and maximum length.

Omission for a new file causes **SYSTEM_SPECIFIED** blocking to be used. For an old file, the preserved value is always used.

- **CHARACTER_CONVERSION**

Specifies whether conversion between the internal character code of a file and ASCII should be performed. The **INTERNAL_CODE** attribute directs conversion if selected.

TRUE

Conversion is performed.

FALSE

No conversion is performed.

Omission for a new file causes **FALSE** to be used.

- **COLLATE_TABLE_NAME**

Specifies the name of a collation table for a keyed file with collated keys. This attribute is ignored for a sequential or byte-addressable file.

For further information, see the SCL Advanced File Management manual.

- **COMPRESSION_PROCEDURE_NAME**

Specifies the name of the optional compression procedure used with the file.

This attribute is ignored for a sequential or byte-addressable file. For more information, see the CYBIL Keyed-File and Sort/Merge Interfaces manual.
- **DATA_PADDING**

Specifies the percentage of space within each new data block of an indexed-sequential file that is to be left unused during initial file creation. This attribute is ignored for a direct access, sequential, or byte-addressable file.

Omission for a new keyed file causes 0 (zero) to be used. For an old keyed file, the preserved value is always used.

For further information, see the SCL Advanced File Management manual.
- **DYNAMIC_HOME_BLOCK_SPACE**

Reserved.
- **EMBEDDED_KEY**

Specifies whether the primary key values of a new keyed file are part of the record data.

This attribute is ignored for a sequential or byte-addressable file.

For further information, see the SCL Advanced File Management manual.
- **ERROR_EXIT_PROCEDURE_NAME**

Specifies the name of an externally declared (XDCL) CYBIL procedure to which control is given whenever an abnormal status is returned by certain file access routine requests. This attribute is equivalent to the **ERROR_EXIT_NAME** attribute and can be used interchangeably.

Omission causes no error exit procedure to be used.
- **ERROR_LIMIT**

Specifies the maximum number of recoverable (nonfatal) file errors that can occur before a fatal error is returned. This attribute is ignored for a sequential or byte-addressable file.

For details, see the SCL Advanced File Management manual.

- **ESTIMATED_RECORD_COUNT**

Specifies your optional estimate of the maximum number of records to be stored in the new file. This attribute is used to calculate a suitable block size for the keyed file.

This attribute is ignored for a sequential or byte-addressable file.

For details, see the SCL Advanced File Management manual.

- **FILE_ACCESS_PROCEDURE_NAME**

Specifies the name of an externally declared (XDCL) CYBIL procedure that intervenes in the calling sequence between users of the file and the file access routines.

Omission for a new file causes no file access procedure to be used. Omission for an old file causes the preserved procedure name to be used.

- **FILE_CONTENTS**

Specifies the type of data contained in the file. It is used by NOS/VE facilities to verify correct usage of a file. Options are:

UNKNOWN

Content is unknown.

OBJECT

Object module or object library.

LIST

Character data for printing that includes a print format effector character as the first character of each record. You cannot specify LIST for a keyed file. If you do, an error is returned when the file is opened.

LEGIBLE

Character data.

ASCII_LOG

Log file in ASCII format.

BINARY_LOG

Log file in binary format.

FILE_BACKUP

Backup file.

SCREEN

Screen formatting form file.

name

Any name other than those indicated in the preceding list.

Omission for a new file causes UNKNOWN to be used. For an old file, the preserved value is always used.

- **FILE_LABEL_TYPE**

Specifies the label type for an ANSI-labelled tape file. The following ANSI label standards are supported:

ANSI 1969 standard - READ only

ANSI 1978 standard - level 1

ANSI 1978 standard - level 2

ANSI 1983 standard revision - level 4

Valid options:

LABELLED (L)

Specifies an ANSI standard label.

UNLABELLED (UL)

Specifies that the tape is unlabelled.

If this attribute is omitted for a new file, UL is assumed. If omitted for an old file, the preserved value is used.

- **FILE_LIMIT**

Specifies the maximum length of the file in bytes. An abnormal status is generated if this limit is exceeded.

NOTE

NOS/VE imposes a default limit of 150,000,000 bytes. It is possible, however, for your site to change this value.

If the length of a keyed file reaches its FILE_LIMIT value, you must enter the COPY_KEYED_FILE command to reinstate the file.

Omission for a new file causes 150,000,000 to be used. For an old file, the preserved value is always used.

- **FILE_ORGANIZATION**

Specifies the organization of a file. A sequential file may be associated with a disk device, magnetic tape, or terminal. A byte-addressable or keyed file can reside only on a disk device. Options are:

- SEQUENTIAL (SQ)
- BYTE_ADDRESSABLE (BA)
- INDEXED_SEQUENTIAL (IS)
- DIRECT_ACCESS (DA)

IS and DA are for keyed files only. Omission for a new file causes SEQUENTIAL to be used. For an old file, the preserved value is always used.

- **FILE_PROCESSOR**

Specifies the name of the processor of the file. This attribute qualifies the FILE_CONTENT attribute. It is used by NOS/VE facilities to verify correct usage of a file. Options are:

- ADA
- APL
- ASSEMBLER
- BASIC
- C
- COBOL
- CYBIL
- DEBUGGER
- FORTRAN
- LISP
- PASCAL
- PL1
- PPU_ASSEMBLER
- PROLOG
- SCL
- SCU
- UNKNOWN
- VX

Omission for a new file causes UNKNOWN to be used. For an old file, the preserved value is always used.

- **FILE_STRUCTURE**

Specifies the structure of the file. This attribute qualifies the **FILE_CONTENT** and **FILE_PROCESSOR** attributes. It is used by **NOS/VE** and its facilities to verify correct file usage.

Options are:

UNKNOWN

The structure is unknown.

DATA

Data file.

LIBRARY

Library file.

name

Name other than **UNKNOWN**, **DATA**, or **LIBRARY**.

Omission for a new file causes **UNKNOWN** to be used. For an old file, the preserved value is always used.

- **FORCED_WRITE**

Specifies whether modified blocks of a file are to remain in memory without being forced to the device when the modification to each block has completed.

This attribute is used only for keyed files and for **US** blocked tape files.

For further information, see the **SCL Advanced File Management** manual.

- **HASHING_PROCEDURE_NAME**

Specifies the optional, user-defined hashing procedure used only for a direct-access file. This attribute is ignored for a sequential or byte-addressable file. The default is **AMP\$SYSTEM_HASHING_PROCEDURE**.

For details, see the **SCL Advanced File Management** manual.

- **INDEX_LEVELS**

Specifies the target number of index levels for a new indexed-sequential file. The default is 2. This attribute is ignored for direct access, sequential, or byte-addressable files.

For details, see the **SCL Advanced File Management** manual.

- **INDEX_PADDING**

Specifies the percentage of space within each new index block of an indexed-sequential file that is to be left unused during the initial file creation. The default is 0 (zero). This attribute is ignored for direct access, sequential, or byte-addressable files.

For details, see the SCL Advanced File Management manual.

- **INITIAL_HOME_BLOCK_COUNT**

Specifies the number of home blocks to be created when a new direct-access file is first opened. This attribute is required for direct-access files. It is ignored for a sequential or byte-addressable file.

For details, see the SCL Advanced File Management manual.

- **INTERNAL_CODE**

Specifies the internal code in which data is represented in the file. It is used by the file access routines to direct tape conversion. It is also available to utilities or application programs to direct conversion on disk files. The attribute selections are:

A6

NOS 6/12 bit display code (ASCII 128-character set).

A8

CYBER 170 8/12-bit ASCII code (ASCII 128-character set).

ASCII

NOS/VE 7-bit ASCII code right-justified in an 8-bit byte (ASCII 128-character set).

D64

NOS 6-bit display code (CDC 64-character set).

EBCDIC

8-bit EBCDIC tape code.

Omission for a new file causes ASCII to be used. For an old file, the preserved value is always used.

- **KEY_LENGTH**

Specifies the length, in bytes, of the primary key for a new keyed file. **KEY_LENGTH** is a required attribute for a new indexed sequential file. This attribute is ignored for a sequential or byte-addressable file.

For an old keyed file, the preserved value is always used.

For details, see the SCL Advanced File Management manual.

- **KEY_POSITION**

Specifies the byte number of each record at which the **EMBEDDED_KEY** field starts. The first byte position is 0 (zero). This attribute is ignored for a sequential or byte-addressable file.

For details, see the SCL Advanced File Management manual.

- **KEY_TYPE**

Specifies how the primary key values of a new indexed-sequential file are compared.

The default is **UNCOLLATED**. This attribute is ignored for direct access, sequential, and byte-addressable files.

For details, see the SCL Advanced File Management manual.

- **LINE_NUMBER**

Specifies the length and location of a line number in each record of a file. The attribute values are specified as:

(location, length)

Line number length is limited to six characters. Line number location is the byte in the line of the beginning of the line number. The first byte in the record has a location of 1.

Omission for a new file indicates the absence of line numbers in the file. For an old file, the preserved value is always used.

- **LOADING_FACTOR**

Reserved.

- **LOCK_EXPIRATION_TIME**

Specifies the number of milliseconds between the time a lock is granted and the time it expires. This attribute is valid only for direct access files. The default is 60,000 milliseconds. This attribute is ignored for sequential and byte-addressable files.

For details, see the SCL Advanced File Management manual.

- **LOGGING_OPTIONS**

Enables the use of keyed-file recovery options. For details, see the SCL Advanced File Management manual, and the CYBIL Keyed-File and Sort/Merge Interface Usage manual.

- **LOG_RESIDENCE**

Specifies the catalog path for the keyed-file's update recovery log. For more information, see the SCL Advanced File Management manual and the CYBIL Keyed-File and Sort/Merge Interfaces manual.

- **MAXIMUM_BLOCK_LENGTH**

Specifies the maximum block length (0 to 2,147,483,615) in bytes. A specification of a maximum block length is ignored when **SYSTEM_SPECIFIED_BLOCKING** is requested (block size is controlled by the operating system in this case). All logical blocks are constrained to **MAXIMUM_BLOCK_LENGTH** or less. Blocks may vary in length between **MINIMUM_BLOCK_LENGTH** and **MAXIMUM_BLOCK_LENGTH**.

This attribute is effective only for record access files. Records are packed into blocks according to ANSI 1978 standards.

For a tape file, this block size determines the maximum size of the physical record written to a tape volume.

For disk files, transfers between central memory and the device are in multiples of one or more blocks.

Omission of block length for a new file causes 4,128 to be used. For an old file, the preserved value is always used.

See the Advanced File Management manual for information on how block lengths are handled for keyed files.

- **MAXIMUM_RECORD_LENGTH**

Specifies the maximum length in bytes (1 to 65,497) allowed for a record. This attribute is used only for files with **RECORD_TYPE = F** or that are keyed files, although certain products (such as Sort/Merge) use the record when processing other record types. F type records are padded to this length on output. This attribute is required for a keyed file, regardless of record type.

Omission for a new file with **RECORD_TYPE = F** causes 256 to be used. For an old file with **RECORD_TYPE = F** or for an old keyed file, the preserved value is always used.

- **MESSAGE_CONTROL**

Specifies which classes of messages are generated during access of a keyed file.

This attribute is ignored for a sequential or byte-addressable file. For details, see the SCL Advanced File Management manual.

- **MINIMUM_BLOCK_LENGTH**

Specifies, in bytes, the minimum block length. This attribute is applicable only for files with `BLOCK_TYPE = USER_SPECIFIED`.

For `RECORD_TYPE = F`, blocks are padded to `MINIMUM_BLOCK_LENGTH` using the `^` character. All blocks are at least `MINIMUM_BLOCK_LENGTH` and do not exceed `MAXIMUM_BLOCK_LENGTH` regardless of record type. The specified value must exceed 17 bytes, which is the length of the longest noise block on tape.

This attribute is ignored for keyed files.

Omission for a new file causes 18 bytes to be used. For an old file, the preserved value is always used.

- **MINIMUM_RECORD_LENGTH**

Specifies the minimum record length in bytes for a new keyed file. This attribute is ignored for a sequential or byte-addressable file.

For details, see the SCL Advanced File Management manual.

- **OPEN_POSITION**

Specifies the positioning to occur when the file is opened.

Options are:

`$BOI`

Position to beginning-of-information.

`$ASIS`

No positioning. See Using the `$ASIS` Open Position earlier in this chapter for more information.

`$EOI`

Position to end-of-information.

If an open position is specified on a file reference, it takes precedence over the file attribute open position.

If a file reference does not specify an open position, the file attribute open position is used.

Omission of the OPEN_POSITION attribute causes \$EOI to be used for the OUTPUT file and \$BOI to be used for all other files.

- PADDING_CHARACTER

Specifies the padding character used to pad short RECORD_TYPE = F records to their MAXIMUM_RECORD_LENGTH. This attribute is used for sequential or byte-addressable files only.

Omission for a new file causes the space character to be used. For an old file, the preserved value is always used.

- PAGE_FORMAT

Specifies the frequency and separation of titling in a legible file. This attribute is used only by the file access routines if the file is associated with a terminal. It is used by other services to prepare files for printing. Options are:

CONTINUOUS (C)

Specifies that a title should appear once at the beginning of the file.

BURSTABLE (B)

Specifies that a title and page number should appear at the top of each page of the file.

NON_BURSTABLE (NB)

Specifies that title and page number should be separated from other data by a triple space rather than forcing top of form as in the burstable selection.

UNTITLED (U)

Specifies that no titling or pagination should appear in the file.

Omission for a new terminal file causes CONTINUOUS to be used. Omission for a new nonterminal file causes BURSTABLE to be used. For an old file, the preserved value is always used.

- **PAGE_LENGTH**

Specifies the number of lines to be written on a printed page. For terminal files, the values of the corresponding terminal attribute is used as the default. For old files, the preserved attribute is always used. For all other files, 60 is used as the default.

- **PAGE_WIDTH**

Specifies the number of characters to be written to a printed line. For terminal files, the values of the corresponding terminal attribute is used as the default. For an old file, the preserved value is always used. For all other files, 132 is used as the default.

- **PRESET_VALUE**

Specifies the integer value to which memory associated with a disk file is initialized. Currently, 0 (zero) is always used.

Omission for a new file causes 0 (zero) to be used. For an old file, the preserved value is always used.

- **RECORD_LIMIT**

Specifies the maximum number of records to be included in a keyed file. This attribute is ignored for a sequential or byte-addressable file. The default is $2^{42}-1$.

For details, see the SCL Advanced File Management manual.

- **RECORD_TYPE**

Specifies the record type. Options are:

FIXED (F)

ANSI fixed length.

TRAILING_CHARACTER_DELIMITED (T)

Records of varying lengths terminated by a single trailing character. The system-supplied record delimiting character is the line-feed character (0A hex).

VARIABLE (V)

CDC variable.

UNDEFINED (U)

Undefined.

Omission for a new record access file causes **VARIABLE** to be used. Omission for other disk files and for keyed files causes **UNDEFINED** to be used. For keyed files, **VARIABLE** and **UNDEFINED** are equivalent. For an old file, the preserved value is always used.

The record types **ANSI-VARIABLE (D)** and **ANSI_SPANNED (S)** are supported only for magnetic tape files. These record types are specified through the **CHANGE_TAPE_LABEL_ATTRIBUTES** command.

- **RECORDS_PER_BLOCK**

Specifies an estimate of the number of data records that are contained in each data block of a new keyed file. This attribute is ignored for a sequential or byte-addressable file or an old keyed file.

For details, see the **SCL Advanced File Management** manual.

- **STATEMENT_IDENTIFIER**

This attribute is applicable to files maintained by the **Source Code Utility (SCU)** and is used to specify the length and location of a statement identifier in each record of the file. The values of the attribute are specified as:

(location,length)

Statement identifier length is limited to 17 characters. Statement identifier location is the byte in the record of the beginning of the statement identifier. The first byte in the record has a location of 1.

This attribute is used only for sequential and byte-addressable files.

Omission for a new file indicates the absence of statement identifiers in the file. For an old file, the preserved value is always used.

- **USER_INFORMATION**

Specifies a string of 32 characters of information you supply that is preserved with the file. This information is not interpreted by **NOS/VE**.

Omission for a new file indicates the absence of user information.

Some file attributes can be changed after a file cycle has been created using the `CHANGE_FILE_ATTRIBUTES` command. Table 4-1 illustrates whether and how file attributes can be changed for an old file.

Table 4-1. Changing Attributes of Old Files

Attribute Name	No¹	Yes²	Temporary
ACCESS_MODE			X
AVERAGE_RECORD_LENGTH	X		
BLOCK_TYPE	X		
CHARACTER_CONVERSION	X		
COLLATE_TABLE_NAME	X		
COMPRESSION_PROCEDURE_NAME	X		
DATA_PADDING	X		
EMBEDDED_KEY	X		
ERROR_EXIT_NAME or ERROR_EXIT_PROCEDURE_NAME			X

1. Attributes in this column cannot be changed. Initial values can be established for a new file via the `SET_FILE_ATTRIBUTE` command.

2. Attributes in this column can be permanently changed by a `CHANGE_FILE_ATTRIBUTE` command.

(Continued)

Table 4-1. Changing Attributes of Old Files (Continued)

Attribute Name	No ¹	Yes ²	Temporary
ERROR_LIMIT			X
ESTIMATED_RECORD_COUNT	X		
FILE_ACCESS_PROCEDURE or FILE_ACCESS_PROCEDURE_ NAME		X	
FILE_CONTENT		X	
FILE_LIMIT		X	
FILE_ORGANIZATION	X		
FILE_PROCESSOR		X	
FILE_STRUCTURE		X	
FORCED_WRITE		X	
HASHING_PROCEDURE_NAME	X		
INDEX_LEVEL	X		
INDEX_PADDING	X		
INITIAL_HOME_BLOCK_COUNT	X		

1. Attributes in this column cannot be changed. Initial values can be established for a new file via the SET_FILE_ATTRIBUTE command.

2. Attributes in this column can be permanently changed by a CHANGE_FILE_ATTRIBUTE command.

(Continued)

Table 4-1. Changing Attributes of Old Files *(Continued)*

Attribute Name	No¹	Yes²	Temporary
INTERNAL_CODE	X		
KEY_LENGTH	X		
KEY_POSITION	X		
KEY_TYPE	X		
LINE_NUMBER		X	
LOCK_EXPIRATION_TIME		X	
MAXIMUM_BLOCK_LENGTH	X		
MAXIMUM_RECORD_LENGTH	X		
MESSAGE_CONTROL			X
MINIMUM_BLOCK_LENGTH	X		
MINIMUM_RECORD_LENGTH	X		
OPEN_POSITION			X
PADDING_CHARACTER	X		
PAGE_FORMAT	X		

1. Attributes in this column cannot be changed. Initial values can be established for a new file via the SET_FILE_ATTRIBUTE command.

2. Attributes in this column can be permanently changed by a CHANGE_FILE_ATTRIBUTE command.

(Continued)

Table 4-1. Changing Attributes of Old Files *(Continued)*

Attribute Name	No¹	Yes²	Temporary
PAGE_LENGTH	X		
PAGE_WIDTH	X		
PRESET_VALUE	X		
RECORD_LIMIT		X	
RECORD_TYPE	X		
RECORDS_PER_BLOCK	X		
RING_ATTRIBUTES		X	
STATEMENT_IDENTIFIER		X	
USER_INFORMATION		X	

1. Attributes in this column cannot be changed. Initial values can be established for a new file via the `SET_FILE_ATTRIBUTE` command.

2. Attributes in this column can be permanently changed by a `CHANGE_FILE_ATTRIBUTE` command.

File attributes are available to any program or procedure (that is, an editor, compiler, or the SCL interpreter) that processes a file. These attributes define the file characteristics for all commands that access files.

Establishing File Attributes

When a file cycle is first created, it is assigned a set of default attributes unless others are provided on a `SET_FILE_ATTRIBUTE` command. Any attributes specified on the `SET_FILE_ATTRIBUTE` command override the default attributes when the file cycle is created. Refer to *Displaying File Attributes* later in this chapter for an example showing the default attributes for a new file.

If a file cycle that has been established by `SET_FILE_ATTRIBUTE` is then opened by a command (such `COLLECT_TEXT`), any attributes defined by the command are merged with those supplied with the `SET_FILE_ATTRIBUTE` command. The merged attributes are then preserved with the file cycle.

A permanent file cycle can be created and its file attributes established before the file is opened and any data is written to the file. The following command sequence can be used to prepare the file attributes in advance:

```
/create_file f=$user.file
/set_file_attributes f=$user.file file_contents=list
/detach_file f=$user.file
```

The `SET_FILE_ATTRIBUTE` command establishes the attributes of the file cycle.

NOTE

Only the `PAGE_LENGTH` and `PAGE_WIDTH` are applicable to terminal files. Only attributes applicable to files with a sequential file organization pertain to tape files (except for the `FORCED_WRITE` and `FILE_LABEL_TYPE` attributes).

Attributes for Record Access Files

Record access files are files containing data organized into records. Read and write operations on the file are then performed on a record by record basis (as opposed to the bit by bit operations performed on segment access files). Refer to the *CYBIL Sequential and Byte-Addressable Files* manual for more information on record and segment access files.

The attributes described in this section apply to record access files with the following types of file organization:

- Sequential
- Byte-addressable
- Keyed file (unless otherwise indicated)

Refer to the following for complete information concerning sequential, byte-addressable, and keyed file access:

- FORTRAN Version 1 Language Definition
- FORTRAN Version 2 Language Definition
- COBOL usage manual
- CYBIL Sequential and Byte-Addressable Files manual

- CYBIL Keyed-File and Sort/Merge Interfaces manual
- SCL Advanced File Management manual

The following are the file attributes pertaining to record access files:

BLOCK_TYPE

MAXIMUM_BLOCK_LENGTH

MAXIMUM_RECORD_LENGTH

MINIMUM_BLOCK_LENGTH (does not apply to keyed files)

PADDING_CHARACTER (does not apply to keyed files)

Attributes for Keyed Files

The attributes listed in this section apply only to keyed files (the FILE_ORGANIZATION attribute is INDEXED_SEQUENTIAL or DIRECT_ACCESS). When used with sequential or byte-addressable files, these parameters are ignored.

The following are the file attributes pertaining to keyed files:

AVERAGE_RECORD_LENGTH

COLLATE_TABLE_NAME

COMPRESSION_PROCEDURE_NAME

DATA_PADDING

DYNAMIC_HOME_BLOCK_SPACE

EMBEDDED_KEY

ERROR_LIMIT

ESTIMATED_RECORD_COUNT

FORCED_WRITE

HASHING_PROCEDURE_NAME

INDEX_LEVEL

INDEX_PADDING

INITIAL_HOME_BLOCK_COUNT
 KEY_LENGTH
 KEY_POSITION
 KEY_TYPE
 LOADING_FACTOR
 LOCK_EXPIRATION_TIME
 LOGGING_OPTIONS
 LOG_RESIDENCE
 MESSAGE_CONTROL
 MINIMUM_RECORD_LENGTH
 RECORD_LIMIT
 RECORDS_PER_BLOCK

Changing File Attributes

The `CHANGE_FILE_ATTRIBUTE` command alters certain file attributes for a file cycle that already exists.

When a file cycle is first used (that is, when it is initially opened), file attributes provided by the creator are preserved with the file. This command allows a subset of the preserved attributes to be changed.

This command may be used to alter file attributes of permanent or temporary files. In either case, the file must not be in use (that is, open) within the job. Other restrictions apply when you use this command:

- A permanent file cycle must not currently be attached to another job.
- If the file cycle must be scheduled for job access using the `ATTACH_FILE` command prior to issuing this command, a share mode value of `NONE` is required.
- The user must have `CONTROL` permission and the user's share requirements must be `NONE`.

Displaying File Attributes

The `DISPLAY_FILE_ATTRIBUTE` command displays the attributes of one or more files. Each display entry will include the file name and the list of requested attribute names and their associated values. The `DISPLAY_FILE_ATTRIBUTES` command can also be used to display a description of how the attribute was defined.

The following example displays the initial default attributes for a new file:

```

/set_file_attributes new_file
/display_file_attributes new_file all
Access_Mode           : (read, shorten, append, modify,
                        execute)
Application_Information : none
Average_Record_Length : 1
Block_Type            : system_specified
Character_Conversion  : no
Collate_Table_Name    : none
Compression_Procedure_Name : none
Data_Padding          : 0
Dynamic_Home_Block_Space : no
Embedded_Key          : yes
Error_Exit_Procedure_Name : none
Error_Limit           : 0
Estimated_Record_Count : 0
File_Access_Procedure_Name : none
File_Contents         : unknown
File_Label_Type       : unlabelled
File_Limit            : 43980465111031
File_Organization     : sequential
File_Processor        : unknown
File_Structure        : unknown
Forced_Write          : no
Global_Access_Mode    : (read, shorten, append, modify,
                        execute)
Global_File_Address    : 0
Global_File_Name       : $000000000S0000D19800812T000000
Global_File_Position   : boi
Global_Share_Mode      : none
Hashing_Procedure_Name : (amp$system_hashing_procedure)

```

1. The maximum file size currently enforced is the value of the site-defined system attribute `MAXIMUM_SEGMENT_LENGTH`.

```

Index_Levels           : 2
Index_Padding          : 0
Initial_Home_Block_Count : 1
Internal_Code          : ascii
Key_Length             : 1
Key_Position           : 0
Key_Type               : uncollated
Line_Number            : ("Location" 1, "Length" 1)
Loading_Factor         : 90
Lock_Expiration_Time   : 60000
Logging_Options        : none
Log_Residence          : none
Maximum_Block_Length   : 4128
Maximum_Record_Length  : 256
Message_Control        : none
Minimum_Block_Length   : 18
Minimum_Record_Length  : 0
Open_Position          : $boi
Padding_Character      : ' '
Page_Format            : burstable
Page_Length           : 60
Page_Width            : 132
Permanent              : no
Preset_Value           : 0
Record_Limit           : 4398046511103
Record_Type            : variable
Records_Per_Block      : 65535
Ring_Attributes        : (11, 11, 11)
Size                   : 0
Statement_Identifier   : ("Length" 1, "Location" 1)
User_Information       : none

```


Copying Files

You can copy one file to another by using the `COPY_FILE` command. For example, to copy file `DATA` to file `TEST_DATA`, enter:

```
/copy_file input=data output=test_data
```

The original file is called the input file; the file into which the original is copied or converted is called the output file. The output file structure (for example, the `FILE_ORGANIZATION`, `BLOCK_TYPE`, and `RECORD_TYPE` attributes of the output file) may differ from the corresponding attributes of the input file.

The `COPY_FILE` command performs either a byte-by-byte copy or a record-by-record copy, depending on the attributes of the specified input and output files.

A byte-by-byte copy does not change the physical representation of the file; the resulting output file is an identical copy of the input file.

Although a record-by-record copy changes the physical representation of the file, its logical content remains the same. That is, the contents of individual records in the file do not change, although the means of accessing them may differ because of differing file attributes.

The following considerations apply to copying files:

- The copy terminates when the `COPY_FILE` command encounters an end-of-information (EOI) in the input file. For unlabelled tape files, the copy terminates when a tapemark is encountered. If the input file is empty, the `COPY_FILE` command returns an abnormal status condition. If the input file is at its end-of-information, the exception condition `FSE$INPUT_FILE_AT_EOI` is returned. If an unlabelled tape is at a double tape mark, the exception condition `AME$INPUT_AFTER_EOI` is returned.
- If an unlabelled tape contains sets of data, each followed by a single tapemark, issue the `COPY_FILE` command once for each set of data to obtain a complete copy of the tape file.
- This command does not copy single tapemarks. For more information on copying tape files, see the Tape Management chapter later in this manual.
- If the output file does not exist, the `COPY_FILE` command will create it and place it in the specified catalog.

- Unless a `SET_FILE_ATTRIBUTE` command has been specified for the created output file, this file inherits the file attributes of the input file. The only exception is the ring attributes of the created file, which default to the ring of the caller of the `COPY_FILE` command.
- The `COPY_FILE` command merges the separate `FILE_CONTENTS` and `FILE_STRUCTURE` attribute values into a single `FILE_CONTENTS` value. The following displays the merged `FILE_CONTENTS`' value.

<u>New FILE_CONTENTS</u>	<u>Old FILE_CONTENTS</u>	<u>Old FILE_STRUCTURE</u>
ASCII_LOG	ASCII_LOG	DATA
BINARY_LOG	BINARY_LOG	DATA
DATA	UNKNOWN	DATA
FILE_BACKUP	FILE_BACKUP	DATA
LEGIBLE_DATA	LEGIBLE	DATA
LEGIBLE_LIBRARY	LEGIBLE	LIBRARY
LIST	LIST	DATA
OBJECT_DATA	OBJECT	DATA
OBJECT_LIBRARY	OBJECT	LIBRARY
SCREEN_FORM	SCREEN	FORM
UNKNOWN	UNKNOWN	UNKNOWN
<name>	<name>	UNKNOWN

If the merging truncates the `FILE_STRUCTURE` value, the warning message `FSE$OUTPUT_STRUCTURE_TRUNCATED` is issued.

- In order to use the COPY_FILE command, you must have the following minimum access and share mode permits for the input and output files:

File	Access Mode	Share Mode
Input	READ	READ, EXECUTE
Output - first choice	APPEND, SHORTEN	NONE
Output - second choice	APPEND	NONE

In the following three examples, the first copies the contents of \$USER.PROLOG to \$LOCAL.A; the second copies file \$USER.X to \$OUTPUT, which is the default; and the third appends file \$USER.INFILE to file \$USER.OUTFILE. In the last example, explicit file positioning is specified for file \$USER.OUTFILE.

```
/copy_file input=$user.prolog output=$local.a
/copy_file $user.x
/copy_file $user.infile $user.outfile.$eoi
```

File Organization Combinations

The following table shows the valid file organization combinations for input and output files. If an attempted copy is invalid, the COPY_FILE command returns an abnormal status condition.

Input File	Output File		
	Sequential	Byte-Addressable	Keyed File
Sequential	Valid	Invalid	Valid
Byte-Addressable	Invalid	Valid	Invalid
Keyed File	Valid	Invalid	Valid

Copying Sequential Files to Sequential Files

When one sequential file is copied to another, the following restrictions determine whether the copy operation is supported:

- If a file has a `FILE_CONTENTS` attribute value of `LIST`, it must not have `UNDEFINED (U)` records and system-specified blocking.
- An input file having `U` records and system-specified blocking cannot be copied to a file whose `FILE_CONTENTS` value is `LIST`.
- If either file has a `FILE_CONTENTS` of `LIST` and has `F` records, its `MAXIMUM_RECORD_LENGTH` must be greater than 1.
- If both files have a `FILE_CONTENTS` attribute value of `LEGIBLE_DATA`, their values for the `LINE_NUMBER` and `STATEMENT_IDENTIFIER` attributes must be identical.
- If the `FILE_CONTENTS` attributes of both files are not identical, one of these attributes must have a value of `UNKNOWN`. However, if the `FILE_CONTENTS` value of one of the files is `LIST`, the other's `FILE_CONTENTS` value can be either `UNKNOWN` or `LEGIBLE_DATA`.

If the following conditions are met, the `COPY_FILE` command performs a byte-by-byte copy of one sequential file to another:

- Both files are mass storage files.
- Both files are opened at their beginning-of-information (BOI).
- The size of the input file is less than or equal to the `FILE_LIMIT` value of the output file.
- The output file is attached for both `APPEND` and `SHORTEN` access.
- The following file cycle attributes are identical for both files:
 - `BLOCK_TYPE`
 - `FILE_ACCESS_PROCEDURE`
 - `FILE_CONTENTS`
 - `FILE_ORGANIZATION (SEQUENTIAL)`
 - `RECORD_TYPE`

Even when the `RECORD_TYPE` attribute values are the same, the following additional restrictions apply:

- When both files contain `F` records, the `MAX_RECORD_LENGTH` and `PADDING_CHARACTER` attribute values must be identical.
- When both files contain `TRAILING_CHARACTER_DELIMITED (T)` records, the `RECORD_DELIMITING_CHARACTER` attribute value must be identical.
- When both files contain `U` records and a user-specified `BLOCK_TYPE`, the `MAX_BLOCK_LENGTH` attribute values must be identical.

When the above processing restrictions are met, the `COPY_FILE` command performs the copy operation with the following results:

- If one file has a `FILE_CONTENTS` value of `LIST` and the other does not, format-effectors are either added or removed. Refer to the section on Copying List Files later in this chapter.
- If the output file has `F` records, the `MAX_RECORD_LENGTH` attribute of the output file must have a value as large as the largest input record; otherwise, input records are truncated to the `MAX_RECORD_LENGTH` value of the output file. When this happens, the exception condition `FSE$OUTPUT_RECORDS_TRUNCATED` is returned.
- While reading an unlabelled tape, the system stops copying when it encounters the end of the last volume provided by the `REQUEST_MAGNETIC_TAPE` command or when it encounters a single (embedded) tapemark within a volume.
- While copying an unlabelled or a nonstandard labelled tape file to an unlabelled tape file, single (embedded) tapemarks found in the input file are not copied to the output file.
- If the output file is an unlabelled tape file, the system writes two tapemarks at the completion of the copy operation. Then the system backspaces the volume to a position preceding the two tapemarks.

When copying record-by-record, the `COPY_FILE` command determines, from the block and record types shown in figure 4-3, the working storage length used to read the input file.

Input File	Output File							
	System-specified (SS) Block Type				User-specified (US) Block Type			
	V	T	U	F	U	F	D*	S*
V, SS	3	3	3	2	1	2	1	3
T, SS	3	3	3	2	1	2	1	3
U, SS	2	2	2	2	1	2	1	2
F, SS	4	4	4	2	1	2	4	4
U, US	3	3	3	2	1	2	1	3
F, US	4	4	4	2	1	2	4	4
D, US*	3	3	3	2	1	2	3	3
S US*	3	3	3	2	1	2	1	3

Key for preceding table:

V VARIABLE
T TRAILING_CHARACTER_DELIMITED
U UNDEFINED
F FIXED
D ANSI_VARIABLE
S ANSI_SPANNED

1 The working storage length is the output file's MAX_BLOCK_LENGTH.
2 The working storage length is the output file's MAX_RECORD_LENGTH.
3 The working storage length is the input file's MAX_BLOCK_LENGTH.
4 The working storage length is the input file's MAX_RECORD_LENGTH.
* Supported for labelled tape files only.

Figure 4-3. Block and Record Types

The values determined in figure 4-3 are incremented by 1 when the FILE_CONTENTS attribute of one (and only one) file is LIST. If the MAXIMUM_BLOCK_LENGTH attribute of an ANSI_VARIABLE file is used to determine the value for the working storage length, then this value is decreased by 4.

Copying Sequential Files to Keyed Files

Under NOS/VE, you can perform a record-by-record copy of a sequential file to a keyed file. Byte-by-byte copies of sequential files to keyed files cannot be performed. The following file attribute restrictions apply:

- The keyed file cannot have a `FILE_CONTENTS` attribute value of `LIST`.
- If the input (sequential) file has a `FILE_CONTENTS` value of `LIST`, it cannot have `U` records and system-specified blocking.
- When the sequential file has a `FILE_CONTENTS` value of `LIST` and contains `F` records, its `MAX_RECORD_LENGTH` value must exceed 1.
- If the `FILE_CONTENTS` value of both files is `LEGIBLE_DATA`, the `LINE_NUMBER` and `STATEMENT_IDENTIFIER` values of each file must be the same.
- The `FILE_CONTENTS` values of the two files must conform to one of the following:
 - The `FILE_CONTENTS` attribute of the input file is `LIST` and the keyed file has a `FILE_CONTENTS` attribute of `LEGIBLE_DATA` or `UNKNOWN`.
 - The `FILE_CONTENTS` attributes of both files are identical, or one file has a `FILE_CONTENTS` value of `UNKNOWN`.

When the preceding restrictions are met, the `COPY_FILE` command performs a record-by-record copy with the following results:

- If the output file is opened at its beginning-of-information, the `COPY_FILE` command releases any data previously written on the output file.
- If the output file is not positioned at its beginning-of-information, the data in the output file is retained and the copy operation merges the two files. However, the `COPY_FILE` command adds only those records from the input file that have unique keys. It copies records from the input file beginning at its open position. During the copy operation, if a record from the input file has a key that already belongs to a record on the output file, the `COPY_FILE` command terminates and returns an abnormal status condition.
- When copying a sequential file to a keyed file, the `COPY_FILE` command assumes that each sequential file record has an embedded key whose location is determined by the `KEY_LENGTH` and `KEY_POSITION` attributes of the output file.
- If the output file has nonembedded keys, the value of the `KEY_POSITION` attribute of the input file is assumed to be 0 and the `KEY_LENGTH` attribute value is determined by the output file's `KEY_LENGTH` attribute.
- When copying a sequential file to a keyed file, the `COPY_FILE` command determines the working storage length used to read the sequential file as follows:
 - If the output file has embedded keys, the working storage length used for the sequential file is equal to the `MAX_RECORD_LENGTH` value of the keyed file.
 - If the keyed file has nonembedded keys, the working storage length used for the sequential file is the sum of the values of the keyed file's `MAX_RECORD_LENGTH` and `KEY_LENGTH` attributes. The working storage length is incremented by 1 if the input file has a `FILE_CONTENTS` value of `LIST`.

When the `COPY_FILE` command encounters records longer than the working storage length, it truncates the records and issues a warning-level abnormal status condition.

Copying Byte-Addressable Files to Byte-Addressable Files

The `COPY_FILE` command cannot copy a byte-addressable file to a file that has a different file organization.

The `COPY_FILE` command performs a byte-by-byte copy of one byte-addressable file to another if the following conditions are met:

- Both files are mass storage files.
- Both files are opened at the same byte address.
- The output file is attached for both `APPEND` and `SHORTEN` access.
- The size of the input file is less than or equal to the value of the `FILE_LIMIT` attribute of the output file.
- The following file attributes of both files are identical:

`BLOCK_TYPE`
`FILE_ACCESS_PROCEDURE`
`FILE_CONTENTS`
`FILE_ORGANIZATION` (byte addressable)
`RECORD_TYPE`

Even when the `RECORD_TYPE` attributes are the same, further restrictions apply:

- When both files have `F` records, the `MAX_RECORD_LENGTH` and `PADDING_CHARACTER` attribute values must be identical.
- When both files have `U` records and user-specified `BLOCK_TYPE` attributes, the `MAX_BLOCK_LENGTH` attributes must be identical.

Copying Keyed Files to Keyed Files

The following restrictions apply to requests for copying a keyed file to another keyed file:

- Keyed files cannot have a `FILE_CONTENTS` attribute value of `LIST`.
- If the `FILE_CONTENTS` attribute of both files has a value of `LEGIBLE_DATA`, the values of the `LINE_NUMBER` and `STATEMENT_IDENTIFIER` attributes for each file must be identical.
- If the `FILE_CONTENTS` attributes of both files are not identical, one of the `FILE_CONTENTS` values must be `UNKNOWN`.
- If the input file has embedded keys and the output file has nonembedded keys, the `MAX_RECORD_LENGTH` value of the output file must equal the difference between the input file's `KEY_LENGTH` value and its `MAX_RECORD_LENGTH` value. The `KEY_LENGTH` values of both files must be identical.
- If the input file has nonembedded keys and the output file has embedded keys, the value of the `MAX_RECORD_LENGTH` attribute of the output file must equal the sum of the input file's `KEY_LENGTH` and `MAX_RECORD_LENGTH` values. The `KEY_LENGTH` values of both files must be identical.
- If both the input and output files have the same `KEY_TYPE` attribute (both have embedded or nonembedded keys), each file must have the same `MAX_RECORD_LENGTH` and `KEY_LENGTH` values.

The `COPY_FILE` command performs a byte-by-byte copy of one keyed file to another when the following conditions are met:

- The output file is not currently open within the job.
- Both files are opened at their beginning-of-information (BOI).
- The output file is attached for both `APPEND` and `SHORTEN` access.
- The size of the input file is less than or equal to the `FILE_LIMIT` value of the output file.
- The following file attributes of both files are identical:

`COLLATE_TABLE_NAME`
`COMPRESSION_PROCEDURE_NAME`
`EMBEDDED_KEY`
`FILE_ACCESS_PROCEDURE`
`FILE_CONTENTS`
`FILE_ORGANIZATION` (keyed files)
`HASHING_PROCEDURE_NAME`
`INITIAL_HOME_BLOCK_COUNT`
`KEY_LENGTH`
`KEY_POSITION`
`KEY_TYPE`
`MAX_BLOCK_LENGTH`
`MAX_RECORD_LENGTH`
`MIN_RECORD_LENGTH`
`RECORD_TYPE`
`RECORDS_PER_BLOCK`

If the above conditions are not met, the `COPY_FILE` command performs a record-by-record copy of the requested keyed files.

NOTE

If you are aware that a record-by-record copy of one keyed file to another will occur, it is more efficient to use the `COPY_KEYED_FILES` command for the `COPY` operation.

The copy operation has the following results:

- If the output file is opened at its beginning-of-information, the COPY_FILE command deletes any data previously written on the output file and replaces it with the contents of the input file.
- If the output file is not opened at its beginning-of-information, the data in the output file is retained and the copy operation results in merging of the input and output files. However, when the COPY_FILE command copies records from the input file, it copies only records with unique keys. If it reads a record from the input file, and the record has a key that already belongs to a record on the output file, the copy operation terminates and returns an abnormal status condition.
- If the input file has embedded keys and the output file has nonembedded keys, the characters established for the KEY_LENGTH attribute of each input record are written as the key on the corresponding output record. The KEY_LENGTH attribute of the output file is observed. Unsatisfactory results may occur if the KEY_POSITION value of the input file is nonzero.
- If the input file has nonembedded keys and the output file has embedded keys, each output record is prefixed by the key of its corresponding input record. The KEY_POSITION and KEY_LENGTH attributes of the output file are observed. Unsatisfactory results may occur if the KEY_POSITION value of the output file is nonzero.

When performing a copy operation from one keyed file to another, the COPY_FILE command determines the working storage length used to read the input file as follows:

- If the input file has embedded keys, the working storage length used is the value of the input file's MAX_RECORD_LENGTH attribute.
- If the input file has nonembedded keys, the working storage length used is the sum of the value of the input file's MAX_RECORD_LENGTH and KEY_LENGTH attributes.
- If the output file's values for the EMBEDDED_KEY, KEY_LENGTH, and MAX_RECORD_LENGTH attributes are not the same as the corresponding values for the input file, truncation may occur; no abnormal status condition is returned.

Copying Keyed Files to Sequential Files

The following restrictions apply when the `COPY_FILE` command copies a keyed file to a sequential file.

- The keyed file cannot have a `FILE_CONTENTS` value of `LIST`.
- If the output file has a `FILE_CONTENTS` value of `LIST`, it cannot contain `U` records and system-specified blocking.
- If the sequential file has a `FILE_CONTENTS` value of `LIST` and contains `F` records, its `MAX_RECORD_LENGTH` value must exceed 1.
- One of the following conditions must be met with respect to the `FILE_CONTENTS` attributes:
 - They must be identical for both files.
 - One file must have a `FILE_CONTENTS` value of `UNKNOWN`.
 - It must have a value of `LEGIBLE_DATA` or `UNKNOWN` for the input file, and `LIST` for the output file. The last case is described under Copying List Files in this chapter.
- If the `FILE_CONTENTS` attribute of both files is `LEGIBLE`, the values of each file's `LINE_NUMBER` and `STATEMENT_IDENTIFIER` attributes must also be identical.

NOTE

In the following cases, the `COPY_FILE` command stores keys for the output file that are different from those used in the input file:

- When the `KEY_POSITION` attributes of the input and output files are not the same (applies only to embedded keys).
 - When the `KEY_POSITION` attribute of the output file is nonzero (applies only to nonembedded keys).
 - When the `KEY_LENGTH` attributes of the input and output files are not identical.
-

- If the `OPEN_POSITION` of the output file is `BOI`, the previous content of the file is released and is replaced by the content of the input file.
- The system considers a sequential file to be a file that contains embedded keys whose location is determined by its `KEY_POSITION` and `KEY_LENGTH` attributes. This consideration applies only if this file is copied to another keyed file.
- When copying from a keyed file to a sequential file, the `COPY_FILE` command determines the working storage length used to read the keyed file as follows:
 - If the keyed file has embedded keys, the working storage length used is the input file's `MAX_RECORD_LENGTH` value.
 - If the keyed file has nonembedded keys, the working storage length used is the sum of the input file's `MAX_RECORD_LENGTH` and `KEY_LENGTH` attributes.
 - The value determined above is increased by 1 if the `FILE_CONTENTS` attribute of the output file has a value of `LIST`.

Copying List Files

The following section describes how NOS/VE handles file-copying operations when either the input file or the output file has a `FILE_CONTENTS` value of `LIST`.

Input File Has a `FILE_CONTENTS` Value of `LIST`

If the `FILE_CONTENTS` attribute of the input file has a value of `LIST` and that of the output file is set to `DATA`, `LEGIBLE_DATA` or `UNKNOWN`, the first character (which is the format effector) of each record of the input file is not copied to the output file.

Output File Has a `FILE_CONTENTS` Value of `LIST`

If the `FILE_CONTENTS` attribute of the input file has a value of `DATA`, `LEGIBLE_DATA` or `UNKNOWN` and the `FILE_CONTENTS` attribute of the output file has a value of `LIST`, the following record-by-record conversion is performed; otherwise the system writes a record-by-record copy to the output file without inserting format effectors.

- If the `PAGE_FORMAT` value is set to `CONTINUOUS`, a triple-space format effector ('-') is inserted in the first record of the output file. A single-space format effector is inserted in all the remaining lines of the output file.
- If the `PAGE_FORMAT` value is set to `BURSTABLE` or `NONBURSTABLE`, a page-eject format effector ('1') is inserted at each interval established by the `PAGE_LENGTH` attribute. A single-space format effector is inserted in all the remaining lines of each page of the output file.

NOTE

If the output file has `F` records, the records may be truncated because of the introduction of format effectors. To avoid truncation, choose a `MAXIMUM_RECORD_LENGTH` for the output file that is greater by 1 than it was on the input file.

Comparing Files

You can compare the contents of two files by using the `COMPARE_FILE` command.

This command performs a binary comparison of data on the specified files (file attributes are not compared). When data on the two files does not match, the system writes the position, content, and logical difference to the specified output file.

The contents of the files are compared from the open position of each until end-of-information on the shorter of the two files.

If the files are identical in content, a normal status is returned. If the files are not identical in content, an abnormal status is returned.

The following example creates two files and compares their contents with the `COMPARE_FILE` command. The `COLLECT_TEXT` command is used to create the text file; `EDIT_FILE` is used to change one character in the file so that a comparison will result in errors (`EDIT_FILE` is described in the `NOS/VE File Editor` manual).

```

/colt o=file_1
ct? This is a temporary file
ct? that will be used in a
ct? COMPARE_FILE example.
ct? **
/copy_file i=file_1 o=file_2
/edit_file f=file_2
Begin editing file: $LOCAL.FILE_2
ef/replace_text t='x' nt='z'
COMPARE_FILE ezample.
ef/end
/compare_file f=file_1 w=file_2
  BYTE ADDRESS   FILE WORD      WITH WORD  LOGICAL DIFFERENCE
          96 46494c4520657861 46494c4520657a61 0000000000000200

-- Specified compare error limit exceeded.
   1 compare errors.
--ERROR-- 1 compare errors.

```

The output from the `COMPARE_FILE` command indicates that a comparison error resulted at byte address 96. The entire contents of the relevant words (in hexadecimal) of each file and their logical difference is also shown.

The following example illustrates how to check for a `CLE$COMPARE_ERRORS_DETECTED` condition.

```
/create_variable n=st v=status
/compare_file a b status=st
/if not st.normal then
  if st.condition= ..
    cle$compare_errors_detected then
      :
    else
      :
  ifend
ifend
```

Your commands to process the comparison error.

Your commands to process some other error.

Displaying the Representation of Data in a File

You can display the contents of a file in hexadecimal, ASCII, or both hexadecimal and ASCII formats by using the `DISPLAY_FILE` command.

If you specify ASCII format, the system displays unprintable ASCII characters (such as NUL or ESC) as spaces.

If you specify hexadecimal format, the system converts each hexadecimal digit to the ASCII representation of the hexadecimal digit (that is, 0 through 9 and A through F).

If you specify both hexadecimal and ASCII, the system displays each line in both formats: the hexadecimal version of the file followed by the printable ASCII equivalent. For example, the ASCII numeral 1 is followed by its hexadecimal equivalent, 31. (Appendix C lists the ASCII character equivalents.)

The following example displays the internal contents of the entire V-record file `FILE_1` created for the `COMPARE_FILE` example earlier in this chapter. Only the ASCII format is requested.

```
/display_file i=file_1 f=ascii
BYTE ADDRESS          ASCII
      0              This is a temporar
    32 y file          that will be
    64 used in a      & COMPARE_
    96 FILE example.
```

The following example repeats the `DISPLAY_FILE` operation, but requests both ASCII and hexadecimal formats:

```

/display_file i=file_1 f=(ascii,hex)
BYTE ADDRESS          HEXADECIMAL          ASCII
   0 0000000000001800 00000000001e5468          Th
  16 6973206973206120 74656d706f726172  is is a temporar
  32 792066696c650000 0000000016000000  y file
  48 0000001e74686174 2077696c6c206265  that will be
  64 207573656420696e 2061000000000000  used in a
  80 150000000000261e 434f4d504152455f  & COMPARE_
  96 46494c4520657861 6d706c652e          FILE example.
    
```

The following example displays the following ranges of byte addresses for file `FILE_1`: bytes 3 through 8; bytes 16 through 24; and bytes 56 through 88:

```

/display_file i=file_1 ba=(3..8,16..24,56..88)
BYTE ADDRESS          HEXADECIMAL          ASCII
   3 000000180000
  16 6973206973206120 74          is is a t
  56 2077696c6c206265 207573656420696e  will be used in
  72 2061000000000000 150000000000261e  a          &
  88 43          C
    
```

Managing Remote Files and Catalogs

If your site supports the Permanent File Transfer Facility (PTF), you can access files and catalogs explicitly on any remote NOS/VE and non-NOS/VE system connected to your system. You can also access files implicitly if both the remote and local systems are NOS/VE.

Implicit Remote NOS/VE File Access

If your implicit remote file reference capability is TRUE (refer to Displaying Your User Validation in chapter 3), then you can implicitly access remote files. To implicitly access a remote file, do the following:

1. Provide your user validation information to the remote system using the `CREATE_REMOTE_VALIDATION` command. You only need to do this once for each remote system you want to access during your job.
2. Use SCL commands to create, copy, and delete files and catalogs implicitly on the remote NOS/VE system. You can also display catalogs.

Creating a Remote Validation

To create a validation on a remote system, use the `CREATE_REMOTE_VALIDATION` command. The validation you create remains in effect until you either log out or delete the validation with the `DELETE_REMOTE_VALIDATION` command.

The following example establishes a remote access to family SKY on a remote system for a user with user name MIKE_B, password STARS, and family name SKY:

```
/create_remote_validation location=sky validation='login ..  
../lu=mike_b pw=stars lf=sky'
```

Use the `DISPLAY_REMOTE_VALIDATION` command to display the names of the remote systems for which you have created a validation during the current job.

If you want to delete a remote validation, use the `DELETE_REMOTE_VALIDATION` command. If you want to replace a remote validation with a different one, use the `DELETE_REMOTE_VALIDATION` command followed by the `CREATE_REMOTE_VALIDATION` command.

Using Commands for Implicit Remote NOS/VE File Access

Once you have created a validation for a remote system, you can use the following commands to access remote files implicitly:

<code>CHANGE_CATALOG_ENTRY</code>	<code>DELETE_CATALOG_PERMIT</code>
<code>CHANGE_FILE_ATTRIBUTE</code>	<code>DELETE_FILE</code>
<code>COPY_FILE</code>	<code>DELETE_FILE_PERMIT</code>
<code>CREATE_CATALOG</code>	<code>DISPLAY_CATALOG</code>
<code>CREATE_CATALOG_PERMIT</code>	<code>DISPLAY_CATALOG_ENTRY</code>
<code>CREATE_FILE</code>	<code>DISPLAY_FILE_ATTRIBUTE</code>
<code>CREATE_FILE_PERMIT</code>	<code>SET_FILE_ATTRIBUTE</code>
<code>DELETE_CATALOG</code>	

You can use the `DISPLAY_CATALOG` command to find out which catalogs and files are available to you on the remote system.

You can specify only one reference to a remote family per command; you cannot specify a remote file for the `OUTPUT` parameter of `DISPLAY` commands. For example, suppose you want to copy the remote file `MY_FILE` to your current mainframe. Then, assuming `MY_FILE` is under family `SYSTEM_B` and username `ME`, you can copy the file by doing the following:

1. If you have not already done so, define a remote validation for family `SYSTEM_B`. You do this by entering the following command:

```
/create_remote_validation l=system_b ..
../v='login lu=me pw=my_password lf=system_b'
```

2. Implicitly reference the file in a `COPY_FILE` command. You do this by entering the following:

```
/copy_file i=:system_b.me.my_file o=$user.file_to_copy_to
```

Explicitly Accessing Remote Files

If your family administrator has authorized you to explicitly access remote files by setting the explicit remote file reference capability to TRUE (refer to Displaying Your User Validation in chapter 3), then you can use the `MANAGE_REMOTE_FILES` command. (You can also use `MANRF` to explicitly access files of different families on the same mainframe.)

With the `MANAGE_REMOTE_FILES` command, you delimit a set of commands to be submitted as a batch job on a remote system. When transferring files between two NOS/VE systems, you can use the two special file transfer commands, `SEND_FILE` and `RECEIVE_FILE`, to transfer files between systems. If you transfer a file to an existing file, the same rules that govern the copy operations of the `COPY_FILE` command determine whether and how the file transfer occurs.

If an error occurs during execution of a file transfer command, the job aborts; no further commands are executed unless you used the `STATUS` parameter on the command in error.

The `MANAGE_REMOTE_FILES` command allows you to access files on a non-NOS/VE system. The format, however, of the command or commands needed to provide remote validation and to perform file transfers varies among the different systems. You may specify the validation commands using the `CREATE_REMOTE_VALIDATION` command. Refer to the Remote Host Facility Usage manual for descriptions of the commands accepted by the supported non-NOS/VE systems.

The examples in this section assume that the remote system is NOS/VE.

When you use MANRF to access a NOS/VE system, follow these guidelines:

- The first command in the set of commands must be a LOGIN command to provide validation information to the remote system. (If you have already created a remote validation for the system with the CREATE_REMOTE_VALIDATION command, you need not include the LOGIN command.)
- After the LOGIN command, you may enter NOS/VE commands. Two additional commands are available with MANRF: SEND_FILE which transfers a file from the remote system to your system; and RECEIVE_FILE which transfers a file from your system to a remote system.
- Only one file can be transferred per set of commands delimited by MANRF.
- End the MANAGE_REMOTE_FILES command list with the character specified by the UNTIL parameter of MANRF. (The default value is two asterisks (**)). Once you end the list of commands, the collection of commands is submitted to the remote system as a batch job.

Transferring a Remote File to Your System

To transfer a remote NOS/VE file to a file on your system, follow these steps:

1. If necessary, establish the file attributes for the file to which you are going to transfer the remote file by using the `SET_FILE_ATTRIBUTES` command.
2. Enter the `MANAGE_REMOTE_FILES` command with the name of the remote system and the name of the file to which you want to copy the remote file. For example:

```
/manage_remote_files location=sky file=$local.temp_file_copy  
mrf?
```

3. If you have not created a remote validation using the `CREATE_REMOTE_VALIDATION` command, enter the `LOGIN` command with the validation information for the remote site. For example:

```
mrf? login login_user=mike_b password=stars login_family=sky
```

4. To transfer the remote file to your system, enter the `SEND_FILE` command. For example:

```
mrf? send_file file=$user.temp_list  
mrf?
```

5. To end the `MANAGE_REMOTE_FILE` session, enter two asterisks after the `mrf?` prompt:

```
mrf?**  
/
```

The system submits the commands you delimited as a batch job on the remote system.

Transferring a File to a Remote System

To transfer a file from your system to a remote system, follow these steps:

1. Enter the `MANAGE_REMOTE_FILES` command with the name of the remote system and the name of local file you want to transfer to the remote system. For example:

```
/manage_remote_files location=sky file=local_text
mrf?
```

2. If you have not created a remote validation using the `CREATE_REMOTE_VALIDATION` command, enter the `LOGIN` command with the validation information for the remote system. For example:

```
mrf? login lu=mike_b pw=stars lf=sky
```

3. If necessary, establish the file attributes for the file to which you want to transfer the remote file by including a `SET_FILE_ATTRIBUTES` command.
4. Enter the `RECEIVE_FILE` command with the name of the remote file to which you want to copy the local file. For example:

```
mrf? receive_file f=$user.text_copy
mrf?
```

5. To end the `MANAGE_REMOTE_FILE` session, enter two asterisks after the `mrf?` prompt:

```
mrf? **
/
```

The system submits the commands you delimited as a batch job on the remote system.

Common Remote File Management Problems

This section discusses some problems that commonly occur when permanent file transfers are being performed.

If your user prolog incurs an error, the PTF server will be not be able to execute SCL commands for you. In addition, no information on the error will be returned to you. Rather, the job log for the serving job is printed on the remote system.

The most common way for a user prolog to incur an error is for a prolog that does not test for interactive jobs. Since the serving job is running as a batch job on the remote system, this can cause an error to be generated in some cases. For information on testing for interactive and batch jobs in your prolog, see the Job Management chapter later in this manual.

When you access a remote file (either implicitly or explicitly), that file is attached for use on the remote system until the access operation is complete. Thus, it is possible that you could initiate another file transfer operation on a remote file that is already attached. This results in an error. In order to avoid this problem, make sure any access operations for a given file have completed before attempting to attach the file for another access operation.

Also, note that the LOCATION parameter on MANRF must be of type string or name even though the DISPLAY_COMMAND_INFORMATION indicates that this parameter may be of type any.

Command and SCL Procedure Execution 5

Command Lists	5-2
Command List Entries	5-3
Object Library Command List Entries	5-3
Catalog Command List Entries	5-4
\$SYSTEM Command List Entry	5-4
Command Utility Command List Entries	5-5
Control Statements Command List Entry	5-5
Control Commands List Entry	5-6
Command List Search Modes	5-7
GLOBAL Search Mode	5-7
RESTRICTED Search Mode	5-7
EXCLUSIVE Search Mode	5-7
Changing the Command List	5-8
Adding Entries to the Command List	5-8
Deleting Entries from the Command List	5-11
Moving Entries within the Command List	5-11
Changing the Search Mode	5-12
Displaying the Command List	5-12
Displaying Command List Entries	5-13

When you tell NOS/VE to execute a command or SCL procedure you are making a *command reference*. The following are the various ways to make a command reference:

- Specify just the name of the command or procedure to be executed if it exists in your command list.

For instance, if you wanted to display the contents of your catalog, simply enter:

```
/display_catalog
```

Since the command exists in your command list, it will be executed.

- Specify just the name of the file containing the command or procedure to be executed if the file resides in a catalog that exists in your command list.

For instance, suppose you have a temporary file named BEAN that contains an SCL procedure named COUNTER. Since \$LOCAL is automatically placed in your command list, to execute procedure COUNTER, you would enter:

```
/bean
```

- Specify the full file path reference to the file containing the command or procedure to be executed.

For instance, suppose user PAT has placed file BEAN in subcatalog CATALOG_1, and that BEAN contains procedure COUNTER. Since CATALOG_1 is not in PAT'S command list, in order to execute procedure COUNTER, PAT would have to enter:

```
/.pat.catalog_1.bean
```

Note that if file BEAN was an object library or module, procedure COUNTER would also have to be included in the reference as follows:

```
/.pat.catalog_1.bean.counter
```

You may also execute a command asynchronously within your job using EXECUTE_COMMAND. Asynchronous tasks and EXECUTE_COMMAND are discussed in the Job Management chapter later in this manual.

Command lists are discussed in detail in the following section. For information on writing SCL procedures, see *Writing SCL Procedures and Command Utilities* later in this manual. For information on executing programs, see the *NOS/VE Object Code Management manual*.

Command Lists

A *command list* is a list of the entire set of commands and functions available to you. The commands and functions can exist in many forms within the command list. For example, commands can reside in object libraries, catalogs, as command entries in \$SYSTEM, or as commands subordinate to a command utility. \$SYSTEM, and each object library and catalog that exists within the command list is known as a *command list entry*.

NOS/VE automatically establishes a command list for each user as part of the user validation and job initialization process. The default command list consists of all the NOS/VE system commands (known as the \$SYSTEM command list entry) and the \$LOCAL catalog. Thus, all local files with the correct file attributes (see *Catalog Command List Entries* later in this chapter for more information) are available as command names.

The SCL command list contains one or more command list entries. The order in which these are listed determines, in part, how NOS/VE searches these entries for commands and functions.

You can change the command list to establish your own command environment. For example, you can add commands to the list and delete or replace any entries in the command list, including the NOS/VE system commands. You can also control the order in which the command list is searched for entries.

Command List Entries

A command list entry is one of the following:

- An object library.
- A catalog.
- The special entry `$SYSTEM`.
- A command utility.

In addition, there are two command list entries that you cannot alter or manipulate:

- The control statements entry, which contains statements such as `IF/IFEND` and is searched before all other command list entries.
- The control commands entry, which contains commands such as `LOGIN` and `LOGOUT` and is searched after all other command list entries.

Object Library Command List Entries

A file used as a command list entry must be an object library. An object library is a file that can contain commands and other modules. A command can be an SCL procedure, a program description, or the starting procedure of an executable program.

For a command to be found in an object library, the command name must be one of the names of an SCL command procedure, one of the names of a program description, or the name of the starting procedure of an object module.

In addition to commands, the message modules on an object library in the command list may be searched for status and help messages. These object libraries may also contain screen formatting modules. See the `NOS/VE Object Code Management` manual for more information on object libraries and their contents.

If you want to overwrite or detach an object library that is in the command list, you must first delete the library from the command list using the `DELETE_COMMAND_LIST_ENTRY` command.

Catalog Command List Entries

A catalog used as a command list entry causes the system to search in the specified catalog for a command. For a command name to be found in a catalog entry, the command name must be the name of a file in the catalog and the file must contain an executable program or an SCL procedure. The FILE_CONTENTS, FILE_STRUCTURE and FILE_PROCESSOR attributes of the file are used to determine how to process the command. The following table illustrates how these attributes are used. Attempting to use a file whose attributes do not match an entry in this table as a command results in an error.

File Content	File Processor	File Structure	Action
OBJECT	Ignored	DATA or LIBRARY	Program is executed.
LEGIBLE or UNKNOWN	SCL or UNKNOWN	DATA or UNKNOWN	SCL Procedure is interpreted.

NOTE

The use of catalogs as command list entries should be avoided in situations in which speed is a concern. Object libraries provide more flexibility than catalogs and can be processed much more efficiently.

\$SYSTEM Command List Entry

\$SYSTEM represents the command list entry containing the commands that are provided by NOS/VE. To execute a command explicitly in the \$SYSTEM command list entry, use:

```
$system.command_name
```

This causes the system to search only the \$SYSTEM command list entry for command_name.

Command Utility Command List Entries

The subcommands and functions provided by a command utility constitute a command list entry. When you start a utility, the system adds the utility command list entry to the beginning of the command list. This makes all of the utility's commands and functions available to you. When you stop a utility, the system removes the utility command list entry from the command list.

Control Statements Command List Entry

Control statements are statements used to structure and control the flow of a job. Control statements belong to a special command list entry that is always searched first. This entry cannot be removed from or replaced in the command list. The following commands also reside in this command list entry:

COLLECT_TEXT

JOB/JOBEND

TASK/TASKEND

UTILITY/UTILITYEND

These commands are also in the \$SYSTEM entry. Refer to SCL Command Streams and Condition Processing later in this manual for a list of control statements.

Control Commands List Entry

A small number of commands provide capabilities basic to SCL and belong to a special command list entry that is always searched last. This entry cannot be removed from or replaced in the command list.

The commands in this group are the following:

ACCEPT_LINE
CHANGE_COMMAND_SEARCH_MODE
CREATE_COMMAND_LIST_ENTRY
CREATE_VARIABLE
DELETE_COMMAND_LIST_ENTRY
DELETE_VARIABLE
DISPLAY_COMMAND_INFORMATION
DISPLAY_COMMAND_LIST
DISPLAY_COMMAND_LIST_ENTRY
DISPLAY_VALUE
INCLUDE_COMMAND
INCLUDE_FILE
INCLUDE_LINE
LOGIN
LOGOUT
PUT_LINE

These commands are also in the \$SYSTEM entry.

Command List Search Modes

The command list can be searched in one of three ways: GLOBAL, RESTRICTED, or EXCLUSIVE.

GLOBAL Search Mode

When GLOBAL search mode is in effect, all entries in the command list can be searched for a command. Also, commands that are specified by path name and command name can be executed.

RESTRICTED Search Mode

With RESTRICTED search mode, all entries in the command list are candidates for being searched. However, in order for a search to proceed beyond the first entry in the command list, the command you enter must be preceded by a slant (/). As with GLOBAL search mode, commands that are specified by path name and command name can be executed.

EXCLUSIVE Search Mode

When EXCLUSIVE search mode is in effect, only the entry at the beginning of the command list is searched for a command. In this mode, commands that are specified by path name and command name are not allowed. In addition, the CHANGE_COMMAND_SEARCH_MODE, CREATE_COMMAND_LIST_ENTRY, and DELETE_COMMAND_LIST_ENTRY commands are not allowed.

SCL assignment and control statements are always available and cannot be removed from the command list.

Changing the Command List

With the following commands, you can add or delete command list entries, specify where a command should be put in the command list, and alter the search mode:

`CREATE_COMMAND_LIST_ENTRY`

Adds entries to either the beginning or the end of the command list.

`DELETE_COMMAND_LIST_ENTRY`

Deletes entries from the command list.

`CHANGE_COMMAND_SEARCH_MODE`

Changes the search mode for the command list.

These commands are described later in this chapter.

Adding Entries to the Command List

If a file or catalog fits the criteria described previously, you can add it to the front or the back of the command list by using the `PLACEMENT` parameter on the `CREATE_COMMAND_LIST_ENTRY` command. Omission of the `PLACEMENT` parameter causes the entry to be placed at the front of the command list.

NOTE

To add an object library to the command list, you must have at least `EXECUTE` access to the object library.

Adding an executable file to the command list, allows you to execute any program or procedure residing on that file by entering its name.

For example, assume that file `DATE_PROCEDURE` contains the following procedure:

```
proc display_date,disd (
    format,f : key month, mdy, dmy, iso, ordinal = month
    status)
    display_value $date($value(format))
procend display_date
```

If this procedure is not added to the command list, it can be executed only by entering the file name. To reference a procedure by the defined procedure names, you must add the procedure to the command list.

When you refer to an entry in the command list, the following rules apply:

- The procedure name can be the same as another name in the command list. However, when the command list is searched for the command name, the first command in the list having the specified name is executed.
- When two commands have the same name but reside in different places in the command list, you can execute the one that resides further down the list by specifying its path name. For example, suppose you have added to the front of your command list a library containing a command that has the same name as a system command. To execute the system command, you need only specify the following:

```
$system.command_name
```

- You can also use the slant character (/) to execute a command that resides further down the command list. The slant causes the first entry in the command list to be skipped when the command list is searched. For example, the Source Code Utility (SCU) has a command called DISPLAY_LIBRARY, with abbreviation DISL. This is the same as the abbreviation for the system command DISPLAY_LOG. From within SCU, the DISPLAY_LOG command could be called as follows:

```
sc//disl
```

Before you can add the file containing the defined procedure(s) to your command list, you must first create an object library. The complete facilities for maintaining object libraries are described in the NOS/VE Object Code Management manual. However, the following examples show you the commands you need to create an object library from a file of one or more procedures.

Assume that a file named `PROCEDURE_FILE` exists in your master catalog and has the following contents:

```
PROC display_date,disd (
    format,f : key month, mdy, dmy, iso, ordinal = month
    status)
    display_value $date($value(format))
PROCEND display_date
PROC display_time,dist (
    format,f : key ampm, hms = ampm
    status)
    display_value $time($value(format))
PROCEND display_time
```

The file contains two procedures: one for displaying the current date and one for displaying the current time.

To create a procedure library and add it to your command list, perform the following steps:

1. Enter:

```
/create_object_library
```

This command initiates the `CREATE_OBJECT_LIBRARY` command utility, which issues the following prompt:

```
COL/
```

2. Subsequent commands can include the `CREATE_OBJECT_LIBRARY` subcommands (refer to the `NOS/VE Object Code Management` manual). Enter the following subcommand to add the procedure file to the object library being created.

```
COL/add_modules library=$user.procedure_file
```

3. Enter the following subcommands to create the object library and terminate the utility session.

```
COL/generate_library library=$user.command_library
COL/quit
```

The `GENERATE_LIBRARY` subcommand generates an object library named `PROCEDURE_LIBRARY`.

4. Add the object library to the command list with the following command:

```
/create_command_list_entry entry=$user.command_library
```

5. After you have entered this command, you can call the procedures directly, as shown in the following examples:

```
/display_time
6:38 PM
/display_date iso
1985-03-28
/dist format=hms
18:38:14
```

Deleting Entries from the Command List

You can delete any entries from the command list by using the `DELETE_COMMAND_LIST_ENTRY` command. For example, to delete the object library added in the preceding example, enter:

```
/delete_command_list_entry entry=$user.command_library
```

Moving Entries within the Command List

To move an entry to the front or end of the command list, delete the entry, then add the entry and specify whether you want the entry to be added at the beginning or end of the list. For example, if you want to move the `$$SYSTEM` entry to the beginning of the command list, enter:

```
/delete_command_list_entry entry=$system
/create_command_list_entry entry=$system
```

With the `$$SYSTEM` entry at the front of the command list, system commands take precedence over local files with the same name.

Changing the Search Mode

You can change the search mode by using the `CHANGE_COMMAND_SEARCH_MODE` command. The `SEARCH_MODE` parameter specifies the new search mode to be associated with the command list. For example:

```
/change_command_search_mode search_mode=restricted
```

This example sets the search mode to `RESTRICTED`.

Displaying the Command List

You can display information about the current command list and/or the search mode governing the command list with the `DISPLAY_COMMAND_LIST` command.

The following is an example of a command list display.

```
/display_command_list display_option=all  
SEARCH MODE IS global  
ENTRIES ARE :$local, $system,  
             :$system.$system.scu.command_library.5
```

Displaying Command List Entries

You can display information about one or more entries in a command list with the `DISPLAY_COMMAND_LIST_ENTRY` command. The following is an example of a portion of a full command list entry display.

```
/display_command_list_entry entry=all display_option=all

Control Statements

block / blockend
cancel
collect_text                colt
continue
cycle
exit
exit_proc
for / forend
if / elseif / else / ifend
job / jobend
loop / loopend
pop
proc / procend
push
push_commands
repeat / until
task / taskend
utility / utilityend
when / whenend
while / whilend

ENTRY :$local

-- Potential commands within a catalog are not shown.
-- No functions in this catalog.
ENTRY $system
Commands

accept_line                  accept_lines, acc1
accept_ntf_messages          accnm
activate_family_administrator
activate_job_statistic       activate_job_statistics, actjs
activate_system_administrator
activate_system_statistic    activate_system_statistics, actss
administer_mail              admm
administer_recovery_log      admr1
administer_scheduling        adms
administer_validations       admv
afterburn_object_text        afterburn_binary, aftot, aftb
analyze_dump                 anad
analyze_object_library        anaol
analyze_program_dynamics     anapd
analyze_resource_usage        anaru
```

Displaying Command List Entries

```
apl
assemble
attach_file          attf
attach_job          attj
backup_log_repositories
backup_permanent_files backup_permanent_file, bacpf
:
submit_job          subj
tdu_post_processor
terminate_job       terminate_jobs, terj
terminate_output    terminate_outputs, terminate_print,
                    terminate_prints, tero, terp
terminate_task
transfer_file_xmodem traf, xmodem
upgrade_software    upgs
vector_fortran      ftn2, vecf, vfortran, vftn, fortran2
vx
wait
wait_for_system_idle waifsi
```

ENTRY \$system

Functions

```
$access_mode
$catalog
$char
$command_source
$condition_code
$condition_name     $condition
:
$value_kind
$variable
$name
```

Control Commands

```
accept_line         accept_lines, acc1
change_command_search_mode chacs
create_command_list_entry create_command_list_entries, crecle
create_variable     create_variables, crev
delete_command_list_entry delete_command_list_entries, delcle
delete_variable     delete_variables, delv
display_command_information discp, display_command_parameter,
                    display_command_parameters, disci
display_command_list disc1
display_command_list_entry discle
display_value       display_values, disv
include_command     incc
include_file        incf
include_line        incl
login
logout
put_line           put_lines, putl
set_command_list    setcl
```

Overview of NOS/VE Jobs	6-1
Interactive Jobs	6-1
Batch Jobs	6-1
Parent and Child Jobs	6-2
Parent and Child Tasks	6-2
Job Names	6-2
Job Classes	6-3
Job Class Restrictions	6-4
Selecting a Job Class	6-4
Job Class Example for Batch Jobs	6-5
Job Class Example for Interactive Jobs	6-7
Job Environments	6-8
Setting Up Prologs and Epilogs	6-9
Job Attributes	6-12
Managing Job Attributes	6-12
Summary of Job Attributes	6-12
Submitting Batch Jobs	6-25
Using SUBMIT_JOB	6-26
Submitting Jobs to Remote Systems	6-26
Using JOB/JOBEND	6-29
Using NOS and NOS/BE ROUTE	6-30
Using ROUTE_JOB	6-31
Terminating Jobs	6-32
Displaying Job Status	6-32
Managing Tasks	6-33
Executing Tasks	6-33
Creating Tasks	6-34
Using TASK/TASKEND	6-35
Using EXECUTE_COMMAND	6-35
Displaying Task Status	6-35
Displaying Active Tasks	6-37
Defining the Primary Task	6-37
Terminating a Task	6-37
Managing Job Output	6-38
Using PRINT_FILE	6-39
Printing Files at CDCNET Batch Stations	6-40
Printing Files at Remote and Partner Systems	6-41
Displaying and Changing Output Attributes	6-44

Summary of Output Attributes	6-45
Displaying Output File Status	6-52
Deleting Files from the Output Queue	6-52
Wait Queue Usage	6-52
Format Effectors	6-53
Terminals	6-53
Using Format Effectors with COPY_FILE	6-54
Using Format Effectors with PRINT_FILE	6-56
Using Standard and Job Files	6-57
Creating File Connections	6-59
Displaying File Connections	6-61
Deleting File Connections	6-61
File Connection Considerations	6-61
Managing Job Logs	6-64
Changing the Message Level	6-65
Changing the Message Language	6-65
Putting Messages into a Job Log	6-65
Putting Messages into an Active Job Log	6-66
Putting Messages into a Job History Log	6-66
Putting Messages into the Display Message Area	6-66
Displaying the Current Job Log	6-67
Displaying a Job History Log	6-69
Displaying Output History	6-69
Displaying Job Resource Limits	6-70
Setting Multiprocessing Options	6-71
Setting Job Sense Switches	6-72

This chapter gives an overview of NOS/VE job concepts; it explains the ways to initiate, manage, and terminate jobs, and how to display the status of jobs including any files created by jobs.

The use of certain system files, called *standard files*, which are used by jobs to direct a job's input and output processing, are described near the end of this chapter

Overview of NOS/VE Jobs

A *job* is a set of tasks executed for a user. A task can be the execution of a command, a utility, a program, or a user-defined task. For more information about task execution, see the NOS/VE Object Code Management manual.

There are two modes of job execution:

- Interactive
- Batch

Interactive Jobs

You begin an interactive job by logging in to NOS/VE from an interactive terminal. You then enter commands after the system prompt. The system prompts you again after the command has been processed. By default, all input to the job is taken from your terminal, and all job output is sent to your terminal. You end the job by logging out.

Because you are interacting with NOS/VE during the job, you can initiate and control the execution of commands, tasks, and other jobs.

Batch Jobs

In batch mode, you submit a job to the system for execution. The system then executes the job as one unit. When a batch job terminates, all job output is, by default, sent to the output queue for printing.

You can submit a batch job from a NOS/VE interactive job, from another NOS/VE batch job, from another operating system (such as NOS or NOS/BE), from a remote NOS/VE or non-NOS/VE system, or from a batch input device. You can also submit jobs to other systems for execution by those systems.

Parent and Child Jobs

The terms parent and child describe the relation of jobs to one another in the system. A *parent job* is a job that initiates the execution of another job; the job that has been initiated by the parent job is called the *child job*.

A parent job can initiate the execution of many child jobs. A child job, in turn, can be the parent job of other child jobs.

Parent and Child Tasks

The terms parent and child are also used to describe the relation of tasks to one another within a job. A *parent task* is a task that initiates the execution of another task; the initiated task is called the *child task*. Child tasks, in turn, can initiate other child tasks.

Job Names

NOS/VE assigns a unique name to each job in the system. This name, called the system-supplied job name, uniquely identifies a job throughout its lifetime.

The system-supplied job name has the following format:

model_serial_alphacounter_integercounter

model	Four-digit model number of the machine. For example, a model 855 would appear as 0855.
serial	Four-digit serial number of the machine.
alphacounter	Three-character alphabetic counter beginning with AAA. Each system maintains its own alphabetic counter.
integercounter	Four-digit integer counter beginning with 0001. Each system maintains its own decimal counter.

The following is an example of a system-supplied job name:

`$0860_0452_AAA_4000`

To specify a system-supplied job name in a command, you can enter either the last two counters preceded by a dollar sign (\$) or, in most cases, just the integer counter preceded by a dollar sign (\$). In the preceding example, the following job names are equivalent:

`$AAA_4000` or `$4000`

The model and serial number components of the job name are added by the system when the value is processed. If you enter just the integer counter, the system will insert the current alpha counter. If the alphabetic counter has not advanced since the job was created (that is, if AAA has not advanced to AAB), the system will insert the correct value for the alphabetic counter. If the alphabetic counter has advanced since the job was created, you must enter both the alphabetic and the integer counter to ensure that you are referencing the correct job name.

In addition to the system-supplied job name, you can give the job a user-supplied name. You can then use either the system-supplied name or the user-supplied name in commands that require you to specify a job name.

Job Classes

Each job in the system is assigned a job class. By default, NOS/VE uses five job classes:

- INTERACTIVE
- BATCH
- SYSTEM
- MAINTENANCE
- UNASSIGNED

Your site administrator can create additional job classes for you.

Every job that enters the system is assigned to a job class. The *job class* assignment controls the operation of the job during its life in the system. For example, the job class determines the initiation priority of the job relative to other jobs and the amount of system resources the job can use during execution. In addition, based on the job class, the system assigns attribute values to the job.

Job Class Restrictions

It is possible for your site to restrict specific jobs from assignment to a particular job class. For this reason, your job may be restricted to a set of job classes since a job may belong to a job class only if it is validated to do so. You can obtain the job classes for which you are validated by using the ADMINISTER_VALIDATION utility discussed in chapter 3.

In addition to the classes permitted for your use, your site administrator can further restrict your access to these job classes through the use of your job's input attributes. A job's input attributes give information used to determine the class in which a job can run in. The following are the input attributes defined for every job entering the system:

CPU_TIME_LIMIT	LOGIN_USER
JOB_MODE	MAGNETIC_TAPE_LIMIT
JOB_QUALIFIER	MAXIMUM_WORKING_SET
LOGIN_ACCOUNT	ORIGINATING_APPLICATION_NAME
LOGIN_FAMILY	SRU_LIMIT
LOGIN_PROJECT	USER_JOB_NAME

All of the previous attributes are discussed in Job Attributes later in this chapter. Your site may choose to use none, some, or all of the previous attributes to determine what job class a job may hold membership in.

Selecting a Job Class

Your site can select a default job class for your interactive or batch jobs. You can determine what this default is by using the ADMINISTER_VALIDATION utility discussed in chapter 3.

If you do not wish to use your default job class, you may specify the name of another job class on the JOB_CLASS parameter of the JOB, SUBMIT_JOB, or LOGIN commands. However, you must be permitted to use the job class you specify, and your job's input attributes must match the qualifications for membership in the job class.

If your job is not allowed membership in the job class you specify, the job is not accepted and an error message is displayed on your screen. For batch jobs, an error message is also placed in the job log of the submitting job.

Some sites may choose to utilize an automatic job class assignment. Automatic job class assignment is requested by specifying the job class name `AUTOMATIC` on the `JOB_CLASS` parameter of the `JOB`, `SUBMIT_JOB`, or `LOGIN` commands. As is the case with any job class, you must be permitted you to use the job class `AUTOMATIC`.

The system assigns membership to the first job class it finds that allows membership based on your job class validation and your job's input attributes. If no class is found meeting this criteria, an error status is issued and the job is denied access to the system.

NOTE

Your site administrator may decide to remove a job class from your system while one of your jobs belonging to that job class is waiting to be executed in the job input queue. When this happens, the job is temporarily assigned to the job class `UNASSIGNED`. The disposition of jobs belonging to job class `UNASSIGNED` is determined by your site.

Job Class Example for Batch Jobs

Suppose your site has peak operating hours between 8 a.m. and 7 p.m. and that in order to most efficiently use its resources, your site personnel have created the following job classes with the following restrictions:

Job Class	Input Attribute	Limits
<code>SMALL_JOB</code>	<code>CPU_TIME_LIMIT</code>	1 .. 20
	<code>MAXIMUM_WORKING_SET</code>	20 .. 300
<code>BIG_JOB</code>	<code>CPU_TIME_LIMIT</code>	21 .. unlimited
	<code>MAXIMUM_WORKING_SET</code>	301 .. unlimited

Further suppose that your site will allow job class SMALL_JOB to be used at any time, but that job class BIG_JOB can be used only from 7 p.m. to 8 a.m. Finally, let SMALL_JOB and BIG_JOB be the only two job classes available for your batch jobs.

In this situation, the following allows MY_JOB membership in job class SMALL_JOB and the job can run at any time:

```
/submit_job f=my_job cpu_time_limit=20 ..  
../job_class=small_job maximum_working_set=300
```

The following allows job NEXT_JOB membership in job class BIG_JOB and the job runs only from 7 p.m. to 8 a.m.:

```
/submit_job f=next_job cpu_time_limit=unlimited ..  
../job_class=big_job maximum_working_set=unlimited
```

Notice that NEXT_JOB may still be queued at 8 a.m. when BIG_JOB can no longer be used. If this happens, NEXT_JOB is temporarily placed in job class UNASSIGNED and your site can then either delete the job or let it execute later in the day.

NOTE

A job class can not be deleted from the system if a job is currently initiated in that job class.

If you are validated to use the automatic job class assignment, the following places your job into job class SMALL_JOB:

```
/submit_job f=newest_job cpu_time_limit=10 ..  
../job_class=automatic maximum_working_set=150
```

The following example, however, results in an error because a job class does not exist for the job:

```
/submit_job f=no_job cpu_time_limit=15 ..  
../job_class=automatic maximum_working_set=500  
--ERROR-- The attributes of this job prevent it from  
being a member of any Job Class.
```

Job Class Example for Interactive Jobs

Suppose your site has a limited number of tape drives. As a result, the following job classes are created at your site:

Job Class	Input Attribute	Limits
INTERACTIVE_DEFAULT	MAGNETIC_TAPE_LIMIT	0
TAPE_JOB	MAGNETIC_TAPE_LIMIT	10

Using these job classes, by default you are placed in job class INTERACTIVE_DEFAULT, and you can not use a tape drive during the course of your interactive job. However, if you do need to use tape drives, you can specify job class TAPE_JOB when you log in. For instance, suppose you are user SUEO, your password is GIPPER, and your family name is VE1. Then, if you are using NAMVE/CDCNET, the following login sequence will place your interactive job in job class TAPE_JOB:

```
User: sueo pw=gipper lf=ve1 mt1=5 jc=tape_job
```

If your site supports automatic job class assignment, the following will also place you in job class TAPE_JOB:

```
User: sueo pw=gipper lf=ve1 mt1=6 jc=automatic
```

The following, however, results in an error:

```
User: sueo pw=gipper lf=ve1 mt1=15 jc=automatic
Error: The attributes of this job prevent it from being a
member of any Job Class.
Incorrect validation entered.
Please try again.
```

```
User:
```

SUEO may then correct the log in information, or disconnect from the network.

NOTE

In general, on a NAMVE/CDCNET connection you can enter any information during your interactive login that you can using the LOGIN command. However, if you wish to enter more than USER, FAMILY, and PASSWORD information, all login values must be entered at the USER prompt.

For any connection other than NAMVE/CDCNET, you can supply only the information for which you are prompted.

Job Environments

A job's environment determines what capabilities the job will have, and how you will interact with it. The following elements make up a job's environment:

- Connection, job, link, program, and terminal attributes.
- Interaction style.
- Working catalog.
- Message mode and natural language.
- Job limits.
- Remote validations.
- Command list entries.
- Job Statistics.
- File connections.
- SCL interpreter options as supplied by the CHANGE_SCL_OPTIONS command.

Most of the above topics are discussed in this manual. Program attributes, however, are discussed in the NOS/VE Object Code Management manual. Also, job statistics are discussed in the NOS/VE System Performance and Maintenance manual, Volume 1, and in the CYBIL System Interface manual.

A prolog is a convenient way to set up values for these elements. Prologs are discussed next.

Setting Up Prologs and Epilogs

A *prolog* is a file containing SCL statements that are executed each time a job is initiated. An *epilog* is a file containing SCL statements that are executed each time a job ends. The particular prologs and epilogs that are executed whenever you begin or end a job depend upon what system you log in to, and what validation is supplied to the system when you log in. Your site may maintain a prolog or epilog for each of the following:

- System

The system prolog or epilog is executed every time you log in or log out of a particular mainframe.

- Job Class

The job class prolog or epilog is executed every time you begin or end a job belonging to a particular job class.

- Account

The account prolog or epilog is executed every time you begin or end a job belonging to a particular account number.

- Project

The project prolog or epilog is executed every time you begin or end a job belonging to a particular project number.

In addition to the prologs and epilogs maintained by your site, you may create prolog and epilog files of your own. These are known as your user prolog and your user epilog. Note that your user prolog is executed after every other site maintained prolog has been executed, and your user epilog is executed before any other site maintained epilog has been executed.

You may use your user prolog and epilog to execute commands that you would normally do everytime you logged in to the system, or just before you logged out of the system. The prolog in particular is useful for setting up your job's environment.

Typically, the name of your prolog file is \$USER.PROLOG, and the name of your epilog file is \$USER.EPILOG.

The following example uses the COLLECT_TEXT command to put the commands to set up your user environment into file \$USER.PROLOG. The following example assumes you are using a DEC VT220 terminal to log in to NOS/VE through NAMVE/CDCNET.

```
/collect_text output=$user.prolog
ct? change_terminal_attribute terminal_model=dec_vt220
ct? change_interactive_style screen
ct? set_working_catalog $user
ct? **
/
```

Since the above example puts the commands into the prolog file, they will be executed each time you log in.

It is often necessary to test for the job mode of the current job since your prolog and epilog files execute everytime you begin and end a job. This is done using an IF/IFEND block (discussed in Command Streams later in this manual) along with the \$JOB function. For instance, the following can be used to execute commands only if the current job is interactive:

```
IF $STRING($JOB(JOB_MODE)) = 'INTERACTIVE' then
    .
    .
    statement list
    .
    .
IFEND
```

You may also want to place in your prolog the commands used to set up screen mode terminate or pause breaks. See Interactive Sessions in this manual for more information.

The names for your prolog and epilog files are assigned by your family administrator. If you want to set up a prolog or epilog, you must create files with those names. However, you can change the names of the prolog and epilog files using the ADMINISTER_VALIDATION utility.

1. To use this utility, enter the ADMINISTER_VALIDATION command; then enter the CHANGE_USER subcommand to initiate the CHANGE_USER utility as follows:

```
/administer_validation
ADM/change_user
CHAU/
```

2. To change the name of your prolog file, use the CHANGE_USER_PROLOG subcommand. To change the name of your epilog file, use the CHANGE_USER_EPILOG subcommand. For instance, to change the name of your epilog file to \$USER.LAST_THINGS_TO_DO, enter

```
CHAU/change_user_epilog '$user.last_things_to_do'
CHAU/
```

3. If you want to display the current name of your prolog and epilog files, use the DISPLAY_USER_PROLOG and DISPLAY_USER_EPILOG subcommands. For example, suppose your user name is SARETT. To display the name of your user prolog, enter the following subcommand:

```
CHAU/display_user_prolog
```

The system displays:

```
SARETT
  USER_PROLOG
  Value: $USER.PROLOG
CHAU/
```

4. To exit from the CHANGE_USER subutility and the ADMINISTER_VALIDATION utility, use the QUIT subcommand as follows:

```
CHAU/quit
ADMV/quit
/
```


Job Attributes

When a job enters NOS/VE, it is assigned a set of job attribute values based on the way the job was submitted and on the job class it was assigned. Your site can change default job attributes for a given job class. Some job attributes affect the manner in which the job is processed; others provide default attribute values for output files generated by the job.

Managing Job Attributes

The following commands are used to manage job attributes:

DISPLAY_JOB_ATTRIBUTES

Displays the current values of a job's attributes.

CHANGE_JOB_ATTRIBUTES

Changes the value of one or more of the current job's attributes.

DISPLAY_JOB_ATTRIBUTES_DEFAULTS

Displays the default job attributes for the current system.

You may also use the `$JOB_DEFAULT` function to return the various default job attributes for the current system. For example, to obtain the default value of the cpu time limit attribute for your system, enter the following:

```
/display_value $job_default(cpu_time_limit)
```

Summary of Job Attributes

The NOS/VE job attributes descriptions follow. Included in the job attribute descriptions are the job attribute default values as well as whether the job attribute value can be changed using the `CHANGE_JOB_ATTRIBUTES` command.

COMMENT_BANNER

Specifies a default 1 to 31 character string that is displayed with the output files generated by the job (including the OUTPUT file). Use of this string is determined by your site.

Can be changed using a `CHANGE_JOB_ATTRIBUTES` command.

CONTROL_FAMILY

Specifies the family name of the control user.

CONTROL_USER

Specifies the user name of the control user. For most jobs, the control user is the same as the login user. However, there are two exceptions to this:

- Jobs entered through a **SUBMIT_JOB**, **DETACH_JOB**, or **JOB/JOBEND** command inherit the user name of the control user who initiated the parent job. Thus, in a chain of submitted jobs, the first job in the chain is the control user of all of the following jobs even though they may all have different login users.
- Jobs entered through a private input/output station have the private station operator assigned as the control user.

Both control users and login users may manipulate their jobs using the **DISPLAY_JOB_STATUS** and **TERMINATE_JOB** commands. Parent jobs may also issue these commands for any child jobs they directly initiated.

COPIES

Specifies the default number of output file copies to be made; the value can be an integer ranging from 1 to 10.

Can be changed using a **CHANGE_JOB_ATTRIBUTES** command.

CPU_TIME_LIMIT

Specifies the system default value in seconds for the maximum cpu time allocated to the job. If a job selects automatic assignment to a job class and specifies no time limit, then the value of this attribute is used. Initially, the value of this attribute is unlimited for both batch and interactive jobs.

CYCLIC_AGING_INTERVAL

Specifies the time, in microseconds, before the memory manager ages the job's working set.

Can be changed using a **CHANGE_JOB_ATTRIBUTES** command.

DETACHED_JOB_WAIT_TIME

Specifies the number of seconds a job, if detached or disconnected from the terminal session, will remain suspended before the job is terminated.

DEVICE

Specifies a default name that, when combined with the STATION parameter value, identifies the printer to which output files generated by the job are to be sent. Values can be a valid printer name or the keyword AUTOMATIC.

If AUTOMATIC is specified, the system prints the file at any printer that meets the specifications supplied for the EXTERNAL_CHARACTERISTICS and FORMS_CODE attributes. Can be changed using a CHANGE_JOB_ATTRIBUTES command.

DISPATCHING_PRIORITY

Specifies the default dispatching priority assigned to all your tasks. Values are P1 (lowest priority) to P10. If DEFAULT is specified, the dispatching priority table established by the job's service class is reinstated. Attempting to raise the value of this parameter has no effect, even though no error status is returned. Can be changed using a CHANGE_JOB_ATTRIBUTES command.

EARLIEST_PRINT_TIME

Specifies the earliest time when output files generated by the job are to be printed. A value of NONE indicates no restrictions.

EARLIEST_RUN_TIME

Specifies the earliest time when the job is to be initiated. A value of NONE indicates no restrictions.

EXTERNAL_CHARACTERISTICS

Specifies a default string to be used by all output files generated by the current job. This string selects a printer that has the same string assigned as its external characteristics. The actual meaning of this string is defined by the site.

Values can be any string of 1 to 6 characters or the keyword NORMAL. If NORMAL is specified, the system selects a printer that has an EXTERNAL_CHARACTERISTICS value of NORMAL. Can be changed using a CHANGE_JOB_ATTRIBUTES command.

FORMS_CODE

Specifies a default string for each output file generated by the job. This string selects a printer that has the same string assigned as its FORMS_CODE value. The actual strings are site-defined.

Values can be any string of 1 to 6 characters or the keyword NORMAL. If NORMAL is specified, the system selects a printer that has a FORMS_CODE value of NORMAL. If NORMAL is specified when the OUTPUT_DESTINATION_USAGE parameter value is DUAL_STATE, the NORMAL value is equivalent to a string of spaces. Can be changed using a CHANGE_JOB_ATTRIBUTES command.

JOB_ABORT_DISPOSITION

Specifies what should be done with the job if it aborts because of system failure. The keywords are:

RESTART

Job is automatically resubmitted so that it starts over from the beginning.

TERMINATE

Job is discarded.

Can be changed using a CHANGE_JOB_ATTRIBUTES command.

JOB_CLASS

Displays the class of the current job.

JOB_MODE

Displays your job's mode. Returnable values are INTERACTIVE or BATCH.

JOB_QUALIFIER

Displays from 1 to 5 names used to qualify the job. These qualifier(s) may limit a job to a specific job class or set of job classes or mainframes. For instance, a job qualifier of VECTOR could be defined to mean that the job requires vector processors. In this way, jobs specifying the VECTOR qualifier could not be submitted to machines that do not have vector processors.

JOB_RECOVERY_DISPOSITION

Specifies what the active job recovery process should do with the job if there is a system interrupt while the job is executing. The keywords are:

CONTINUE

An attempt is made to reestablish the state of the job as it was at the point of interruption. If the attempt succeeds, the job will continue normal execution. If the attempt fails, the value specified for the JOB_ABORT_DISPOSITION attribute is used to determine disposition of the job.

RESTART

Job is automatically resubmitted so that it starts over from the beginning.

TERMINATE

Job is discarded.

Can be changed using a CHANGE_JOB_ATTRIBUTES command.

JOB_SIZE

Specifies the size in bytes of your job's input file. For interactive jobs, this value is always zero.

JOB_SUBMISSION_TIME

Specifies the time when your job arrived in the input queue.

LATEST_PRINT_TIME

Specifies the latest time that output files created by your job are to be printed. If the output file does not get printed by the latest print time, it is discarded. A value of NONE indicates no restrictions.

LATEST_RUN_TIME

Specifies the latest time that the job may be initiated. If the job does not get initiated by the latest run time, it is discarded. A value of NONE indicates no restrictions.

LOGIN_ACCOUNT

Specifies the account name under which your job is scheduled and run.

LOGIN_FAMILY

Specifies the family name under which your job is scheduled and run.

LOGIN_PROJECT

Specifies the project name under which your job is scheduled and run.

LOGIN_USER

Specifies the user name under which your job is scheduled and run.

MAGNETIC_TAPE_LIMIT

Specifies the maximum number of magnetic tape drives required at one time by the job.

The job may only be a member in a job class that supports a magnetic tape limit greater than or equal to the value of this attribute.

MAXIMUM_WORKING_SET

Specifies the maximum number of pages to be contained in the job's working set. If necessary the memory manager will remove pages from the working set to assure this value is not exceeded. Can be changed using a **CHANGE_JOB_ATTRIBUTES** command.

MINIMUM_WORKING_SET

The **MINIMUM_WORKING_SET** specifies a page aging algorithm threshold. If the number of pages in a working set is less than or equal to the **MINIMUM_WORKING_SET** value, then the pages in that working set are not aged. Normal aging is resumed when the number of pages in the working set exceeds the **MINIMUM_WORKING_SET** value.

Can be changed using a **CHANGE_JOB_ATTRIBUTES** command.

OPERATOR_FAMILY

Specifies the default private station or the default family name of the station operator to whom output files generated by the job are sent. If the OUTPUT_DESTINATION_USAGE value for an output file is PRIVATE or NTF, this family name together with the value of the OPERATOR_USER attribute identifies the private station operator or remote system operator who can print or receive the file. This attribute is also used to establish the CONTROL_FAMILY attribute of output files with OUTPUT_DESTINATION_USAGE values of PRIVATE or NTF.

OPERATOR_USER

Specifies the default private station or remote system operator user name attribute for output files generated by this job. If the OUTPUT_DESTINATION_USAGE value for an output file is PRIVATE or NTF, this user name together with the OPERATOR_FAMILY attribute identifies the private station operator or remote system operator who can print or receive the file. This attribute is also used to establish the CONTROL_USER attribute of output files with OUTPUT_DESTINATION_USAGE values of PRIVATE or NTF.

Can be changed using a CHANGE_JOB_ATTRIBUTES command.

ORIGINATING_APPLICATION_NAME

Specifies the name of the application which entered your job to the system.

OUTPUT_CLASS

Specifies the default output class for output files generated by this job. The output class defines the initial priority, maximum priority, an aging interval, and an aging factor for the output file.

The only defined output class is NORMAL. This means all output files have an initial priority of 100, a maximum priority of 3700, an aging interval of 1 second, and an aging factor of 1 priority unit per aging interval.

Can be changed using a CHANGE_JOB_ATTRIBUTES command.

OUTPUT_DESTINATION

Specifies the default location name of the system where the output file is sent for printing if the file's OUTPUT_DESTINATION_USAGE attribute is QTF or NTF. For all other values of OUTPUT_DESTINATION_USAGE, this attribute is ignored.

A location name is a name associated with a remote system, such as a family name or a logical identifier. Location names are determined by your site.

Can be changed using a `CHANGE_JOB_ATTRIBUTES` command.

`OUTPUT_DESTINATION_USAGE`

Specifies the default for either the kind of CDCNET print station where the file is printed, or the queued-file transfer application used to forward the output file to a remote system. The following options are available:

`PUBLIC`

Indicates the file will be printed at a public CDCNET batch I/O station. When `PUBLIC` is specified, the values of the `OPERATOR_FAMILY`, `OPERATOR_USER`, `OUTPUT_DESTINATION`, and `REMOTE_HOST_DIRECTIVE` attributes are ignored.

`PRIVATE`

Indicates the file will be printed at a private CDCNET batch I/O station when the designated station operator is controlling the station. When `PRIVATE` is specified, values of the `OUTPUT_DESTINATION` and `REMOTE_HOST_DIRECTIVE` attributes are ignored.

`DUAL_STATE`

Indicates the file will be printed under control of the partner system. When `DUAL_STATE` is specified, the only other meaningful attribute values are the values of the `FORMS_CODE`, `COPIES`, `ROUTING_BANNER`, and `REMOTE_HOST_DIRECTIVE` job attributes.

`QTF`

Indicates that the file will be forwarded to the remote system identified by the `OUTPUT_DESTINATION` job attribute for processing by that system.

`NTF`

Indicates that the file will be forwarded to a remote NTF system for processing by that system. See your site personnel for more information on NTF.

Can be changed using a `CHANGE_JOB_ATTRIBUTES` command.

OUTPUT_DISPOSITION

Specifies the default disposition of the job's standard output. Allowable values are either a file name or one of several keywords. The following list describes the results of each of the allowable values.

file_name

Specification of a file name causes the standard output to be copied to the specified permanent file at job end. You may not specify a remote family name with this file name. If the attempt to copy the standard output file to this file fails, the output file is sent to the output queue for printing.

DISCARD_ALL_OUTPUT (DAO)

All output generated by the job is discarded. This includes the standard output file at jobend. This option has no effect unless the job destination is a NOS/VE or NTF system.

DISCARD_STANDARD_OUTPUT (DSO)

Standard output is discarded at job end. This option has no effect unless the job destination is a NOS/VE or NTF system.

LOCAL (L)

Any output generated by the job is printed at the destination system rather than being returned to the originating user's default output station.

If the job destination is a NOS/VE system, the destination system's default for OUTPUT_DESTINATION_USAGE is used rather than the job's normal default value.

PRINTER (P)

Any output generated by the job is returned to the originating user's default output station.

WAIT_QUEUE (WQ)

For jobs transferred remote systems, any output generated by the job is returned to the originating user's \$WAIT_QUEUE subcatalog on the originating system, using the user's job name as the file name. If the file already exists, a file cycle which corresponds to \$NEXT is created.

For jobs executing on the local system, the output is placed in the executed job's LOGIN_USER's \$WAIT_QUEUE subcatalog.

If the \$WAIT_QUEUE subcatalog does not exist at the time the output files are returned, it is created for the user.

Can be changed using a CHANGE_JOB_ATTRIBUTES command.

OUTPUT_PRIORITY

Specifies the default increment that is added to the output file's initial priority (defined by the output class) for all output files generated by this job. Values can be:

Keyword	Increment
LOW	0
MEDIUM	1500
HIGH	3000

Can be changed using a CHANGE_JOB_ATTRIBUTES command.

PAGE_AGING_INTERVAL

Specifies the number of CP microseconds that elapses before the memory manager ages the job's working set.

If you need to age your job, and if your job does not go into extended waits or have numerous asynchronous, unrelated tasks, set this attribute to a low value, and the CYCLIC_AGING_INTERVAL attribute to a high value. Otherwise, set the CYCLIC_AGING_INTERVAL attribute to a low value and the PAGE_AGING_INTERVAL to a high value.

Can be changed using a CHANGE_JOB_ATTRIBUTES command.

PURGE_DELAY

Indicates the length of time the output file generated by the job remains in the output queue after it is printed.

REMOTE_HOST_DIRECTIVE

Specifies a default text string which may be used to control output processing of output files, or control processing of jobs submitted to remote systems. How this string is interpreted depends upon the following:

- If this string is intended to control output processing of output files, then this string should contain one of the following:
 - A PRINT_FILE command for output files to be printed on a NOS/VE system.
 - A ROUTE command for output files to be printed on a non-NOS/VE system.
 - The ROUTE command's parameters for output files to be printed on the non-NOS/VE side of a dual-state system.
- If this string is intended to control processing of a job submitted to a remote system, then this string should contain one of the following:
 - A SUBMIT_JOB command for jobs submitted to remote NOS/VE systems for processing.
 - A ROUTE command for jobs submitted to non-NOS/VE systems for processing.

This parameter is ignored unless the OUTPUT_DESTINATION_USAGE output attribute or the JOB_DESTINATION_USAGE parameter on the SUBMIT_JOB command specify the appropriate value. For more information on submitting jobs to remote systems, see Submitting Jobs to Remote Systems later in this manual. For more information on submitting output files to remote and partner systems, see Printing Files at Remote and Partner Systems.

Can be changed using a CHANGE_JOB_ATTRIBUTES command.

ROUTING_BANNER

Specifies a default character string to be displayed with output files generated by this job. The actual use of this string is determined by the site.

Can be changed using a CHANGE_JOB_ATTRIBUTES command.

SENSE_SWITCHES

Displays the setting of the job's sense switches.

SERVICE_CLASS

Displays the name of the job's service class.

SITE_INFORMATION

Displays the **SITE_INFORMATION** string associated with all output files created by your job.

SRU_LIMIT

Specifies the maximum number of System Resource Units (SRUs) allocated to the job. The value must be larger than the user's SRU limit assigned when the user was validated or the job will be rejected. As the job executes if the accumulated SRUs exceed this value, a job abort limit condition will occur and the job will terminate.

The job may only be a member in a job class that supports a SRU limit greater than or equal to the value of this attribute.

STATION

Specifies a default I/O station name (or the control facility name in the case of a private station or NTF remote system) to which the file is to be sent.

Values can be any valid station name or the keyword **AUTOMATIC**. If **AUTOMATIC** is specified, the system default is used.

Can be changed using a **CHANGE_JOB_ATTRIBUTES** command.

SYSTEM_JOB_NAME

Displays the name assigned to your job by the system.

USER_INFORMATION

Specifies a user information string of up to 256 characters. This string enables you to pass information (such as a file path) to a submitted job. This string is also passed on to all output files generated by the submitted job.

Can be changed using a **CHANGE_JOB_ATTRIBUTES** command.

USER_JOB_NAME

Displays the name assigned to the job by the user.

VERTICAL_PRINT_DENSITY

Specifies the default vertical print density at which the file is to be printed. This value will affect the selection of the printer where the file is printed. Select one of the following keywords:

SIX

Selects a printer to print at six lines-per-inch.

EIGHT

Selects a printer to print at eight lines-per-inch.

NONE

Vertical print density is not used to select a printer.

FILE

Vertical print density of the source file is used to determine the print density. If the source file attribute is 6, SIX is used. If the source file attribute value is in the range of 7 through 12, EIGHT is used.

Can be changed using a CHANGE_JOB_ATTRIBUTES command.

VFU_LOAD_PROCEDURE

Specifies the default name of a procedure file containing the definition of a vertical forms unit (VFU) load image that must be loaded into the printer before the file is printed. This attribute affects printer selection.

The keyword NONE indicates that the file need not be printed on a printer capable of using VFU load procedures or that the default VFU load procedure should be used.

Specification of a procedure file causes the system to select a printer capable of using the VFU load procedures and then download the procedure file to the printer before the file is printed.

Can be changed using a CHANGE_JOB_ATTRIBUTES command.

Submitting Batch Jobs

There are several ways you can submit a batch job for execution by NOS/VE:

- Create a file containing a set of SCL statements. The first statement in the file must be the LOGIN command. You can then submit the file for execution as a batch job using the SUBMIT_JOB command. You can issue the SUBMIT_JOB command from an interactive job or from another batch job.
- Delimit a list of commands as a job and automatically submit the job to the system for execution using the JOB/JOBEND control statements. When you submit a job in this manner, the validation information is taken from the parent job so you need not include a LOGIN command. You can use the JOB/JOBEND control statements to delimit a job in a terminal session, within a file, and within a batch job.
- Route jobs to NOS/VE from another system. Jobs can be forwarded to NOS/VE by a queued-file transfer application, such as QTF, or through a dual-state partner system. To route jobs from a NOS/VE system, use a SUBMIT_JOB command. To route jobs from NOS, use the NOS ROUTE command. To route jobs from NOS/BE, use the NOS/BE ROUTE command. To route jobs from other remote host types, use the appropriate MFQUEUE command as described in the Remote Host Facility Usage manual.
- Submit a job to a NOS/VE system from a batch station input device. You can precede the job with the ROUTE_JOB command if you want to specify a user-supplied job name, the system on which the job should run, and the job output destination. You can issue the ROUTE_JOB command as the first image in a card deck.
- Include JOB/JOBEND commands or a SUBMIT_JOB command in a set of SCL commands submitted through the MANAGE_REMOTE_FILES command from another NOS/VE system or submitted through MFLINK from a non-NOS/VE system.

For information about controlling the output produced by a job, refer to Managing Job Output later in this chapter.

Using SUBMIT_JOB

To submit a file as a batch job using the SUBMIT_JOB command, follow these steps:

1. Create a file that contains a series of SCL statements. The first statement in the file must be the LOGIN command. You can end the series of commands with the LOGOUT command; however, including it is optional.

For example, the following sequence of commands shows how to use the COLLECT_TEXT command to create a file to be submitted for batch execution:

```
/collect_text batch_file
ct? login login_user=sdh password=pass456 ..
ct? login_family=nve
ct? display_command_list all
ct? logout
ct? **
```

2. When you want to submit the file as a batch job, use the SUBMIT_JOB command. For example, the following entry submits file BATCH_FILE for execution with the user-supplied job name MY_JOB. The job's output is written to the file \$USER.MY_JOB_OUTPUT.

```
/submit_job file=batch_file user_job_name=my_job ..
../output_disposition=$user.my_job_output
```

Submitting Jobs to Remote Systems

You may use the SUBMIT_JOB command to execute jobs on remote systems if your site has the appropriate queued-file transfer applications connecting your system to a remote system (see your site personnel for more information on your site's configuration).

To execute a job on a remote system, you must specify a location name as the JOB_DESTINATION parameter of the SUBMIT_JOB command. The location name is a name associated with the remote system, such as a family name or a logical identifier. Location names are determined by your site. For more information, see your site administrator.

If you are submitting the job to a remote system, you may also need to specify which queued-file transfer application you are using to transfer the job to the remote system. This is done using the `JOB_DESTINATION_USAGE` parameter on the `SUBMIT_JOB` command. You may specify `QTF`, `NTF`, or some other site-developed queued-file transfer application on this parameter. See your site personnel for more information on the appropriate queued-file transfer application to specify on this parameter. If you have specified a remote system, the default value for the `JOB_DESTINATION_USAGE` parameter is `QTF`.

NOTE

If both the `JOB_DESTINATION` and the `JOB_DESTINATION_USAGE` parameters are omitted from the `SUBMIT_JOB` command, the `LOGIN_FAMILY` job attribute is used to determine to which system the job should be submitted. If the `LOGIN_FAMILY` attribute value is not a local family name, the job is forwarded to `QTF` for transfer to a remote system.

In some cases, you must also use the `REMOTE_HOST_DIRECTIVE` (`RHD`) parameter on the `SUBMIT_JOB` command to submit jobs to remote systems. The `REMOTE_HOST_DIRECTIVE` job attribute specifies a text string that may be used to control output processing of the job. How this attribute is interpreted depends on the value of the `JOB_DESTINATION_USAGE` (`JDU`) parameter of `SUBMIT_JOB`. The following table summarizes the possible interpretations of the `REMOTE_HOST_DIRECTIVE` attribute for each of the possible values of `JDU`:

JDU	RHD
QTF	<p>Specifies the text string to be interpreted by the destination remote host identified by the <code>JOB_DESTINATION</code> job attribute. The required content of the directive string depends on the remote host type:</p> <ul style="list-style-type: none"> For a <code>NOS/VE</code> remote host system, specify a <code>SUBMIT_JOB</code> command with no <code>FILE</code> parameter. For example: <pre> /submit_job file=my_job/jd=ve_family jdu=qtf/rhd='subj odi=local s=station_1'</pre>

JDU	RHD
-----	-----

QTF (cont.)	For transfers between two NOS/VE systems, this attribute is not usually needed because attributes are preserved during transfers. However, if the file is being relayed through a non-NOS/VE system, you must specify the appropriate SUBMIT_JOB command on the RHD attribute or the following attribute values will be lost during the transfer:
----------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

```

CPU_TIME_LIMIT
EARLIEST_RUN_TIME
JOB_EXECUTION_RING
JOB_QUALIFIER
LATEST_RUN_TIME
MAGNETIC_TAPE_LIMIT
MAXIMUM_WORKING_SET
OPERATOR_FAMILY
OPERATOR_USER
SRU_LIMIT
STATION
USER_INFORMATION

```

In addition, the file reference case of the OUTPUT_DISPOSITION attribute may be lost during the transfer.

- If the remote host is a non-NOS/VE system, refer to the Remote Host Facility Usage manual for descriptions of the MFQUEUE remote host directives accepted by the specific systems.

NTF	Depends on the type of the destination system. See your site personnel for more information.
-----	----------------------------------------------------------------------------------------------

Suppose file MY_JOB has the following contents:

```

login login_user=user_abc password=abc_password
display_catalog catalog=$user

```

To submit this job to the destination system MF1, enter:

```

/submit_job file=my_job job_destination=mf1

```

Notice that the JOB_DESTINATION_USAGE parameter has not been specified, so QTF is used by default. Also, since no OUTPUT_DISPOSITION has been specified, the job's output will be returned to your local default output station.

Using JOB/JOBEND

You can delimit and submit a list of one or more SCL statements to be executed as a batch job by using the JOB/JOBEND command. You would normally use JOB/JOBEND during an interactive session to submit a short series of commands for batch mode execution on a one-time basis, or to delimit a job within a procedure.

The batch job created and submitted by JOB/JOBEND inherits the LOGIN_USER, PASSWORD, and LOGIN_FAMILY of the parent job. You need not provide a LOGIN command for validation: NOS/VE generates appropriate LOGIN and LOGOUT commands when it processes JOB/JOBEND.

To submit a batch job interactively using JOB/JOBEND, follow these steps:

1. Enter the JOB command. The system responds with the following prompt:

```
job/
```

2. After the JOB/ prompt, enter a series of SCL statements.
3. To end the list and submit the job for execution, enter the JOBEND command. For example, the following commands direct NOS/VE to compile and execute a CYBIL program:

```
/job
job/cybil i=$user.cybil_program
job/igo
job/jobend
/
```

JOB/JOBEND treats all lines within it as containing only statements, and a line ending with two or more periods is treated as a continuation line. If you try to use commands such as COLLECT_TEXT (or other commands that allow you to enter a series of commands or subcommands or text) you may experience problems. For best results, submit such jobs using the SUBMIT_JOB command.

Note that the JOB command cannot be used to submit a job to a remote system.

Using NOS and NOS/BE ROUTE

To submit a batch job to NOS/VE from a NOS or NOS/BE system, use the ROUTE command.

The NOS and NOS/BE ROUTE commands include parameters that allow you to specify the destination for your job file and the destination for the job's output. The parameters you need are described below:

- The LFN parameter specifies the name of the local file you want to submit as a NOS/VE job.
- The DC parameter specifies an input disposition code for the file named on the LFN parameter. The following input disposition code options are available:
 - DC=IN specifies that the file should be queued with an input disposition.
 - DC=NO specifies that the file should be queued with an input disposition and that output not explicitly routed by the job should be discarded.
 - DC=TO specifies that the file should be queued with an input disposition and that output should be queued with a wait disposition unless explicitly routed elsewhere.
- The ST=lid parameter specifies the logical identifier (lid) of the machine to which you want to route the job.
- The DO=lid parameter specifies the logical identifier (lid) of the machine to which you want to route the job's output.

(For complete documentation of the NOS ROUTE command, refer to volume 3 of the NOS Version 2 Reference Set; for complete documentation of the NOS/BE ROUTE command, refer to the NOS/BE Version 1 Reference Manual.)

The following example submits the NOS file MYJOB to system NVE and causes the job's output to be sent to either the central site or the mainframe from which the job was routed:

```
route,myjob,dc=in,st=nve
```

The following example submits file MYJOB to system NVE and causes the job's output to be placed in the NOS wait queue:

```
route,myjob,dc=to,st=nve
```

The following example routes the file MYJOB to system NVE and causes the job's output to be sent to the user with a terminal ID of AB.

```
route,myjob,dc=in,st=nve,tid=ab
```

Using ROUTE_JOB

When you submit a batch job to NOS/VE from an I/O station input device, you have the option of preceding the job with the ROUTE_JOB command. With ROUTE_JOB, you can specify a user-supplied job name, the system on which the job should run, and the destination for output files.

For card input, you must start the command in column six. The first four columns must contain /*BC which indicates batch command text. Here is an example of a card input job:

```
/*BC route_job, jn=job1, jd=nosve1, jod=public_io_station_1
.
.
input job
.
.
/*EOI
```

If necessary, you can continue the command over a series of cards. For example,

```
/*BC route_job jn=job2 ..
/*BC jd=nosve2 ..
/*BC jod=nosve_control_facility ..
/*BC un=batch_user_1 ..
/*BC uf=nosve2
.
.
input job
.
.
/*EOI
```

Terminating Jobs

The `TERMINATE_JOB` command terminates a job. You can use the `TERMINATE_JOB` command to terminate a job that has been initiated or that is waiting to be initiated (a job cannot terminate itself). However, you can only terminate child jobs and jobs for which you are the login user or the control user.

If you terminate a job that is waiting to be initiated, it is eliminated as a candidate for execution. If you terminate a job that has been initiated, `NOS/VE` causes an abnormal termination. This termination includes releasing all files and resources used by the job and routing of its output file and job log for printing. Conditions established by `WHEN` statements are not processed (see `SCL Command Streams` and `Condition Processing` later in this manual for an explanation of the `WHEN` statement).

Displaying Job Status

To display the status of an active job, use the `DISPLAY_JOB_STATUS` command. For example:

```
/display_job_status job_name=my_job do=all
Control_Family      : nve
Control_User       : sarett
CPU_Time_Used      : Job Mode- 16.161
                   : Monitor Mode- 1.463
Display_Message    : put1 ' hi'
Job_Class          : batch
Job_Destination_Usage : ve
Job_Mode           : batch
Job_State          : initiated
Login_Family       : nve
Login_User         : sarett
Operator_Action_Posted : no
Page_Faults       : Assigned- 484
                   : From Disk- 128
                   : Reclaimed- 154
System_Job_Name    : $0990_0102_aaaj_0722
User_Job_Name      : my_job
```

Managing Tasks

A *task* is the unit of execution within a job. A task can be the execution of a command, a utility, a program, or a user-defined task.

This section discusses how you can define and control the execution of tasks within your job.

Executing Tasks

In NOS/VE, tasks execute either synchronously or asynchronously. With *synchronous* execution, the task initiator waits until the initiated task terminates before it continues executing (the task initiator may be a job or a parent task). With *asynchronous* execution, the task initiator continues executing immediately after initiating the task.

For example, with synchronous execution if TASKA initiates TASKB then TASKA must suspend itself. If TASKB then synchronously initiates TASKC, TASKB must also suspend itself. In this situation, in order for TASKB to resume execution, TASKC must terminate, and in order for TASKA to resume execution, TASKB must terminate. Thus, for synchronous execution, a parent task must wait for all of its child tasks to terminate before it can resume execution.

TASKA, TASK B, TASKC could also execute asynchronously. In this way, each task is executing independently of the other. However, for asynchronous execution a parent task cannot terminate normally until all of its child tasks terminate. In other words, in order for TASKA to terminate normally during asynchronous execution, it must wait for TASKB to terminate. Note, however, that if TASKA terminates abnormally, then TASKB (and all of TASKB's child tasks) is terminated at once.

Whether a task initiates synchronously or asynchronously depends upon how it was created. See the following section for information on how to create tasks.

Creating Tasks

Tasks can be created both implicitly and explicitly. You implicitly create a task by executing programs or most utilities (normally, executing a NOS/VE command that does not start a utility will not create a task). When you implicitly create a task, you are always executing the task synchronously.

You can create a task explicitly using one of the following commands:

TASK/TASKEND

Can create a task that executes either synchronously or asynchronously. See the following section for more information on TASK/TASKEND.

EXECUTE_COMMAND

Executes a single NOS/VE command as an asynchronous task.

EXECUTE_TASK

Executes a single program as either a synchronously or asynchronously executing task. See the Object Code Management manual for details.

Using TASK/TASKEND

You can use `TASK/TASKEND` to delimit a sequence of commands to be executed as a task. If you do not name the task, the task is executed synchronously. If you name the task on the `TASK` command, the task is executed asynchronously.

The following example shows how to use `TASK/TASKEND` to asynchronously execute a task to monitor a job's status every 10 minutes and display a message when the job completes:

```
/task check_job
task/while $strrep($job_status(my_job,job_state))<>'UNKNOWN'
task/wait 10*60*1000 "ten minutes"
task/whilend
task/display_value 'MY_JOB is done'
task/taskend
```

Using EXECUTE_COMMAND

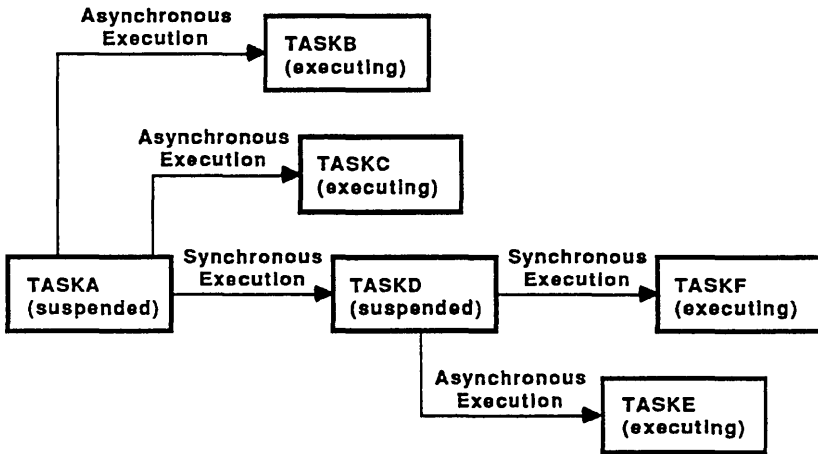
To execute a single command as an asynchronous task, use the `EXECUTE_COMMAND` command, followed by the command name. Note that you cannot execute a command utility with this command.

Displaying Task Status

You can display the current status of one or more asynchronous tasks by using the `DISPLAY_TASK_STATUS` command. A message is displayed informing you that the specified task or tasks are still executing, have completed execution normally, or have been terminated due to an error. For example:

```
/display_task_status tn=mytask
MYTASK terminated with ..
--ERROR AM 1016-- FSP$OPEN_FILE was issued for file,
:$LOCAL.F.1, which does not exist.
```


You can display task status only for tasks initiated by the requesting group of synchronously executing tasks. For example, consider the following group of tasks:



M02213

In the preceding figure, TASKA asynchronously initiated TASKB and TASKC. TASKA then synchronously initiated TASKD which, in turn, asynchronously initiated TASKE. TASKD then went on to synchronously initiate TASKF. In this situation, TASKA, TASKD, and TASKF are the requesting group of synchronously executing tasks and they are the only tasks that can request the status of TASKB, TASKC, and TASKE.

The keyword ALL for the TASK_NAME parameter of the DISPLAY_TASK_STATUS command can be used to display the status for all tasks initiated by the requesting group of synchronously executing tasks.

Displaying Active Tasks

You can display data about active tasks within your job by using the `DISPLAY_ACTIVE_TASKS` command. For example,

```
/disat
ACTIVE TASKS = 2
OCP$CREATE_OBJECT_LIBRARY job time = 0.060 monitor time = 0.016 page faults = 78
$JOBMNTR job time = 0.570 monitor time = 0.148 page faults =311
/
```

Defining the Primary Task

The *primary task* is the task to which break conditions (such as terminate break and pause break) are sent. For tasks which are executing synchronously, the innermost child task is the primary task.

If the tasks within your job are executing asynchronously, a task can issue a `DEFINE_PRIMARY_TASK` command to designate itself as the primary task of the job. If a primary task is terminated, its parent task becomes the new primary task. If the primary task is terminated and it has no parent task, the job is also terminated.

Terminating a Task

The `TERMINATE_TASK` command is used to terminate a task. However, a task can only be terminated by its creator. For instance, suppose `TASKA` created `TASKB`, and `TASKB` created `TASKC`. Then `TASKB` can only be terminated by `TASKA`, and `TASKC` can only be terminated by `TASKB`. At no time can `TASKC` terminate `TASKB` or `TASKA`, nor can `TASKB` terminate `TASKA`. In addition, `TASKA` can not terminate `TASKC` directly. Note, however, that if `TASKA` terminates `TASKB` while `TASKC` is executing, `TASKC` will be terminated as well.

Managing Job Output

For interactive jobs, the standard output produced by the job is sent to the terminal as it is produced.

For batch jobs, the default destination for output produced by the job is file OUTPUT. At the end of job processing, file OUTPUT is queued for printing according to the job's job attributes.

For all jobs, you can use the PRINT_FILE command to queue a file for printing.

There are several ways you can control files queued for printing using the PRINT_FILE command and job attributes:

- When you submit a batch job for execution using the JOB or SUBMIT_JOB command, you can use the OUTPUT_DISPOSITION parameter to specify that the output should be written to the named permanent file or placed in the \$WAIT_QUEUE subcatalog. Using this same parameter, you can also discard standard output, discard all output or specify the location at which the output is to be printed. See the OUTPUT_DISPOSITION parameter on the JOB or SUBMIT_JOB command for details.
- If you want the batch job's standard output to be discarded, you can include a CHANGE_JOB_ATTRIBUTES command in the job. For example:

```
change_job_attributes odi=discard_standard_output
```

- You can change the job attributes that control the attributes of the queued file by including the CHANGE_JOB_ATTRIBUTES command in the job.
- You can change the job attributes that control job output for all your jobs by including the CHANGE_JOB_ATTRIBUTES command in your user prolog. The following example specifies that all batch job output and all files queued for printing in the batch job using the PRINT_FILE command will be routed to the NOS partner system, which will then route them to the mainframe with the logical identifier (LID) M02:

```
if $string($job(job_mode)) = 'BATCH' then
  change_job_attributes odu=dual_state ..
  rhd='dc=pr,ec=a9,st=m02'
ifend
```

- If your output is queued for printing in the NOS/VE system from which the job originated, you can display, change, and terminate the output using the `DISPLAY_OUTPUT_ATTRIBUTES`, `CHANGE_OUTPUT_ATTRIBUTES`, and `TERMINATE_OUTPUT` commands.

The following sections describe job output manipulation in more detail:

Using `PRINT_FILE`

You can schedule one or more files for printing by using the `PRINT_FILE` command. Depending on the value of the `OUTPUT_DISPOSITION` job attribute, the `PRINT_FILE` command causes the system to place the file in the output queue, discard the file, or place the file in the wait queue. If the file is placed in the output queue, the system removes the file from the queue and uses the file's output attributes to transfer the file to one of the following destinations:

- A CDCNET batch station.
- The partner system on a dual-state mainframe.
- A remote NOS/VE system's output queue.
- A non-NOS/VE system.

The wait queue is described later in this section. Parameters on the `PRINT_FILE` command specify a file's output attributes. Default values for parameters not specified on the call to the `PRINT_FILE` command are obtained from your job attributes.

See the *Printing Files at Remote and Partner Systems* section for more information on printing files using remote and dual-state partner systems.

When files are printed with `DATA_MODE=CODED`, the first character of each record (print line) is used as a format effector to control line spacing. For more information, see *Format Effectors* later in this chapter.

The following example illustrates the use of the `PRINT_FILE` command. In this example, the output destination and all other parameters are supplied by default:

```
/fortran i=fortran_source l=fortran_listing
/print_file fortran_listing
```

Printing Files at CDCNET Batch Stations

You may print a file at either a public or private CDCNET Batch Station. To do this, you must make sure the appropriate job attribute values exist for the output file. This can be done using the PRINT_FILE command. The following table summarizes what attributes must be specified in order to print a file at a CDCNET batch station.

Destination	Attribute	Value
Public	OUTPUT_ DESTINATION_USAGE	PUBLIC
	STATION	Station name
Private	OUTPUT_ DESTINATION_USAGE	PRIVATE
	STATION	Station control facility name
	OPERATOR_FAMILY	Station operator's family name
	OPERATOR_USER	Station operator's user name

If you attempt to print a file at a CDCNET batch station and you have supplied incorrect values for the STATION, OPERATOR_FAMILY, or OPERATOR_USER attributes, the output file is placed in the output queue, though it is not printed and no error messages are returned.

Refer to the following section for information on printing files at a CDCNET batch station on a remote system.

Printing Files at Remote and Partner Systems

If you are using a dual-state system, you may use the `PRINT_FILE` command to print files on the partner operating system. You may also use the `PRINT_FILE` command to print files using systems other than the one on which you are currently executing, if your site has the appropriate queued-file transfer applications connecting your system to a remote system. See your site personnel for more information on your site's configuration.

The `OUTPUT_DESTINATION` (ODE) and `OUTPUT_DESTINATION_USAGE` (ODU) attributes of a file specify the file's destination. The following table summarizes the appropriate ODE and ODU values for each of the file's possible destinations. The ODE and ODU attributes may be changed using the `PRINT_FILE`, `CHANGE_JOB_ATTRIBUTES`, and `CHANGE_OUTPUT_ATTRIBUTES` commands.

Destination System	ODE	ODU
Dual-state partner	None (ignored)	DUAL_STATE
Remote NOS/VE	Location name	QTF
Remote non-NOS/VE	Location name	QTF or NTF

The location name value of the `OUTPUT_DESTINATION` parameter is a name associated with a remote system, such as a family name or a logical identifier. Location names are determined by your site. For more information, see your Site Administrator.

The following example illustrates printing a file at the remote system, M09:

```
/print_file file=fortran_listing ..
../output_destination=M09 ..
../output_destination_usage=qtf
```

If you have specified a remote system for the `OUTPUT_DESTINATION` attribute, and the `OUTPUT_DESTINATION_USAGE` attribute is not `QTF`, `NTF`, or some other queued-file transfer application, the output file will not be transferred to the remote system, and it will be placed in the local system's output or wait queue, depending upon the value of the job's `OUTPUT_DISPOSITION` attribute.

You can also use the `REMOTE_HOST_DIRECTIVE` parameter on the `PRINT_FILE` command to control file processing on a remote or dual-state partner system. The value that you specify for this parameter depends on the value of the `OUTPUT_DESTINATION_USAGE` attribute. The following table summarizes the possible values of the `RHD` parameter:

ODU	RHD
PUBLIC	Ignored.
PRIVATE	Ignored.
DUAL_STATE	Indicates that the values specified on this parameter are supplied to the <code>NOS</code> or <code>NOS/BE ROUTE</code> command. The dual-state link attributes indicate the user validation under which the file is routed. The dual-state system creates the <code>'ROUTE,filename'</code> portion of the string for you. For example:

```
/print_file file=data_file odu=dual_state ..  
../rhd='dc=pr,ec=a9,st=m02'
```

Refer to the `NOS Reference Set, Volume 3`, or the `NOS/BE Reference Manual` for descriptions of the `ROUTE` command attributes.

If the values on the `RHD` parameter are incorrect, the `NOS` or `NOS/BE ROUTE` command may fail and you will not receive any message reporting the failure. Refer to `Dual-State File Access` later in this manual for more detailed information on link attributes and dual-state access.

ODU**RHD****QTF**

Indicates the text string interpreted by the remote host identified by the `OUTPUT_DESTINATION` parameter. The content of the directive string depends on the remote host type:

- For a NOS/VE remote host system, specify the RHD parameter on either a `PRINT_FILE` command with no `FILE` parameter, or on a `CHANGE_JOB_ATTRIBUTE` command. For example:

```
/print_file file=data output_destination=nv2 ..
../output_destination_usage=qtf ..
../rhd='prif odu=public s=station_1'
```

For transfers between two NOS/VE systems, this attribute is not usually needed because attribute values are preserved during transfers. However, if the file is being relayed through a non-NOS/VE system, you must specify the appropriate `PRINT_FILE` or the following attribute values may be lost:

```
COMMENT_BANNER
EARLIEST_PRINT_TIME
LATEST_PRINT_TIME
OPERATOR_FAMILY
OPERATOR_USAGE
OUTPUT_DESTINATION_USAGE
PURGE_DELAY
ROUTING_BANNER
STATION
VERTICAL_PRINT_DENSITY
VFU_LOAD_PROCEDURE
```

- If the remote host is a non-NOS/VE system, refer to the Remote Host Facility manual for descriptions of the MFQUEUE remote host directives accepted by specific remote systems.

NTF

Depends on the type of the destination system. See your site personnel for more information.

If you are printing a file on a dual-state partner system and you do not use the `REMOTE_HOST_DIRECTIVE` parameter, the file will be printed on the partner system's local batch printer. The following example directs that any output from the `PRINT_FILE` command be printed on the dual-state system:

```
/change_job_attribute ..  
../output_destination_usage=dual_state
```

However, if the job that contains the `PRINT_FILE` command was issued from a `NOS/VE` remote batch facility station, the resulting output will print at the originating station's printer regardless of the value specified for the `OUTPUT_DESTINATION_USAGE` parameter.

Displaying and Changing Output Attributes

An output file is a file that is scheduled to be written to an output device. An output file's attributes describe and control how the file is printed. The following commands are used to manage a file's output attributes. To use them, you must be logged in to the `NOS/VE` system where the output file was generated and must be the login user or control user for the file.

`DISPLAY_OUTPUT_ATTRIBUTES`

Displays the attributes of an output file.

`CHANGE_OUTPUT_ATTRIBUTES`

Changes the attributes of an output file. If the output file is printing, its attributes cannot be changed.

Summary of Output Attributes

The following are the output attribute names. You may change the value of some of these attributes by using the `CHANGE_OUTPUT_ATTRIBUTES` command.

COMMENT_BANNER

Specifies a character string to be displayed with the printed file. The use of this string is determined by the site.

Can be changed using the `CHANGE_OUTPUT_ATTRIBUTES` command.

CONTROL_FAMILY

Specifies the family name of the control user.

CONTROL_USER

Specifies the user name of the control user. For most jobs, the control user is the same as the control user of the job that generated the output file. The only exception to this is if the output file's destination is a private input/output station. In this situation, the private station operator is assigned as the control user of the output file.

Control users can manipulate output files using the `CHANGE_OUTPUT_ATTRIBUTES`, `DISPLAY_OUTPUT_ATTRIBUTES`, `DISPLAY_OUTPUT_STATUS`, and `TERMINATE_OUTPUT` commands. Login users use these commands to manipulate their own output files.

COPIES

Specifies the number of copies to be printed.

Can be changed using the `CHANGE_OUTPUT_ATTRIBUTES` command.

COPIES_PRINTED

Displays the number of copies that have been made so far.

DATA_MODE

Specifies whether the data is coded or transparent. When files are printed with `DATA_MODE=CODED`, the first character of each record (print line) is used as a format effector to control line spacing. For more information, see Format Effectors later in this chapter.

DEVICE

Specifies a name that, when combined with the STATION attribute value, identifies the printer at which the file is to be printed.

Values for this attribute can be a valid printer name or the keyword AUTOMATIC. If you specify AUTOMATIC, the system prints the file at any printer that meets the EXTERNAL_CHARACTERISTICS and FORMS_CODE specifications.

Can be changed using the CHANGE_OUTPUT_ATTRIBUTES command.

DEVICE_TYPE

Displays the type of device to which the file is scheduled. Currently this value will always be PRINTER.

EARLIEST_PRINT_TIME

Displays the earliest time the file is to be printed. A value of NONE indicates that no restrictions apply.

EXTERNAL_CHARACTERISTICS

Specifies a string that is used to select a printer that has the same string defining its external characteristics. The actual meaning of this string is defined by the site.

Values can be any string of 1 to 6 characters or the keyword NORMAL. If you specify NORMAL, the system selects a printer that has an EXTERNAL_CHARACTERISTICS value of NORMAL.

Can be changed using the CHANGE_OUTPUT_ATTRIBUTES command.

FILE_POSITION

Displays a restarting point for output. If the connection with the output station is lost during output, this attribute is used to record the position at which the transfer was discontinued. Currently, this attribute's value will always be beginning-of-information.

FILE_SIZE

Indicates the size (in bytes) of the output file.

FORMS_CODE

Specifies a string that is used to select a printer that has the same string defining its forms code attribute. The actual meaning of this attribute is defined by the site.

Values can be any string of 1 to 6 characters or the keyword **NORMAL**. If you specify **NORMAL** when the **DESTINATION_USAGE** attribute is **DUAL_STATE**, the **NORMAL** value is equivalent to a string of spaces.

Can be changed using the **CHANGE_OUTPUT_ATTRIBUTES** command.

LATEST_PRINT_TIME

Displays the latest time a file is to be printed. If the file is not printed by this time, it is discarded. **NONE** indicates there are no restrictions.

LOGIN_ACCOUNT

Displays the account name of the job that generated the output file.

LOGIN_FAMILY

Displays the family name of the job that generated the output file.

LOGIN_PROJECT

Displays the project name of the job that generated the output file.

LOGIN_USER

Displays the user name of the job that generated the output file.

OPERATOR_FAMILY

Specifies the family name of a private station or remote system operator. This family name together with the **OPERATOR_USER** attribute identifies the private station operator or remote system operator who can print or receive the file. This attribute is also used to establish the **CONTROL_FAMILY** attribute of the output file. This attribute is not meaningful unless the **OUTPUT_DESTINATION_USAGE** attribute specifies **PRIVATE** or **NTF**.

Can be changed using the **CHANGE_OUTPUT_ATTRIBUTES** command.

OPERATOR_USER

Specifies the user name of a private station or remote system operator. This user name together with the **OPERATOR_FAMILY** attribute identifies the private station operator or remote system operator who can print or receive the file. This attribute is also used to establish the **CONTROL_USER** attribute of the output file. This attribute is not meaningful unless the **OUTPUT_DESTINATION_USAGE** attribute specifies **PRIVATE** or **NTF**.

Can be changed using the `CHANGE_OUTPUT_ATTRIBUTES` command.

`ORIGINATING_APPLICATION_NAME`

Displays the name of the application that entered to the system the job that generated the output file.

`OUTPUT_CLASS`

Specifies an output class for the output file. The output class defines the initial priority, maximum priority, aging interval, and an aging factor for the output file.

The only defined output class is `NORMAL`. This means all output files have an initial priority of 100, a maximum priority of 3700, an aging interval of 1 second, and an aging factor of 1 priority unit per aging interval.

Can be changed using the `CHANGE_OUTPUT_ATTRIBUTES` command.

`OUTPUT_DESTINATION`

Specifies the location name of the system where the output file is sent for printing if the file's `OUTPUT_DESTINATION_USAGE` attribute is `QTF` or `NTF`. For all other values of `OUTPUT_DESTINATION_USAGE`, this attribute is ignored.

A location name is a name associated with a remote system, such as a family name or a logical identifier. Location names are determined by your site.

Can be changed using the `CHANGE_OUTPUT_ATTRIBUTES` command.

`OUTPUT_DESTINATION_USAGE`

Specifies either the kind of `CDCNET` print station where the file is printed, or the queued-file transfer application used to forward the output file to a remote system. The following options are available:

`PUBLIC`

Indicates the file will be printed at a public `CDCNET` batch I/O station. When `PUBLIC` is specified, the `OPERATOR_FAMILY`, `OPERATOR_USER`, `OUTPUT_DESTINATION`, and `REMOTE_HOST_DIRECTIVE` attributes are ignored.

PRIVATE

Indicates the file will be printed at a private CDCNET batch I/O station at a time when a designated station operator is controlling the station. When PRIVATE is specified, the OUTPUT_DESTINATION and REMOTE_HOST_DIRECTIVE attributes are ignored.

DUAL_STATE

Indicates that the file is to be printed under control of the partner system. If this value is specified, the only other meaningful attributes are the values of the FORMS_CODE, COPIES, ROUTING_BANNER, and REMOTE_HOST_DIRECTIVE attributes.

QTF

Indicates that the file is forwarded to a remote system for printing by that system.

NTF

Indicates that the file is forwarded to a remote NTF system for printing by that system. See your site personnel for more information on NTF.

Can be changed using the CHANGE_OUTPUT_ATTRIBUTES command.

OUTPUT_PRIORITY

Specifies a priority increment that is added to the output file's initial priority (defined by the output class). Values can be:

Value	Increment Value
--------------	------------------------

LOW	0
MEDIUM	1500
HIGH	3000

OUTPUT_SUBMISSION_TIME

Displays the time when the job sent the output file to the queue.

PURGE_DELAY

Displays the length of time the file remains in the output queue after it has printed. NONE indicates the file is purged immediately.

REMOTE_HOST_DIRECTIVE

Specifies a default text string which is to control the processing of output files generated by the job. How this attribute is interpreted depends on the value of the OUTPUT_DESTINATION_USAGE (ODU) attribute. See the Submitting Jobs to Remote Systems section or the Printing Files at Remote and Partner Systems section for more detailed information.

Can be changed using the CHANGE_OUTPUT_ATTRIBUTES command.

ROUTING_BANNER

Specifies a character string to be displayed with the printed file. The actual use of this string is determined by the site.

Can be changed using the CHANGE_OUTPUT_ATTRIBUTES command.

SITE_INFORMATION

Displays the SITE_INFORMATION string associated with the job that generated the output file.

STATION

Specifies the I/O station name (or the control facility name in the case of a private station or the NTF remote station) to which the file is sent.

Values can be any valid station name or the keyword AUTOMATIC. If you specify AUTOMATIC, the system default is used.

Can be changed using the CHANGE_OUTPUT_ATTRIBUTES command.

SYSTEM_FILE_NAME

Displays the system-supplied name of the output file. This file is created by the NOS/VE system on which the PRINT_FILE command was entered. The created name is unique (no other file on the network will have the same name).

SYSTEM_JOB_NAME

Displays the system-supplied name of the job that generated the output file. This name is created by the NOS/VE system on which the job was submitted. The created name is unique (no other job on the network will have the same name).

USER_FILE_NAME

Displays the user-supplied name of the output file. If no name is specified, the file's name is used.

USER_INFORMATION

Displays the user information string associated with the job that generated the output file.

USER_JOB_NAME

Displays the user-supplied name of the job that generated the output file.

VERTICAL_PRINT_DENSITY

Specifies the vertical print density at which the file is to be printed. This value will affect the selection of the printer where the file is printed. Select one of the following keywords:

SIX

Selects a printer to print at six lines per inch.

EIGHT

Selects a printer to print at eight lines per inch.

NONE

Vertical print density is not used to select a printer.

Can be changed using the **CHANGE_OUTPUT_ATTRIBUTES** command.

VFU_LOAD_PROCEDURE

Specifies the name of a procedure file containing the definition of a vertical forms unit (VFU) load image that must be loaded into the printer before the file is printed. This attribute affects printer selection.

You can specify the keyword **NONE** to indicate that the file need not be printed on a printer capable of using VFU load procedures or that the default VFU load procedure should be used.

If you specify the name of a procedure file, the system selects a printer capable of using the VFU load procedures and the procedure file is downloaded to the printer before the file is printed.

Can be changed using the **CHANGE_OUTPUT_ATTRIBUTES** command.

Displaying Output File Status

To obtain information about the status of an output file, use the `DISPLAY_OUTPUT_STATUS` command. You can obtain this information only if the file is currently in the output queue of the NOS/VE system where the requesting job is executing. In addition, you can only display the status of output files for which you are login or control user.

Deleting Files from the Output Queue

To delete a file (or files) from the NOS/VE output queue, use the `TERMINATE_OUTPUT` command. You can only delete files that are in the queue and for which you are the login user or the control user. If a file is already being printed, you cannot terminate the printing.

Wait Queue Usage

You can have output from your batch job placed in subcatalog `$WAIT_QUEUE` within your `$USER` catalog. If you specify a value of `WAIT_QUEUE` on the `OUTPUT_DISPOSITION` parameter of the `SUBMIT_JOB` or `JOB` command, output generated by the job is placed in the wait queue under a file whose name is the same as the user's job name. For example, output from the following job will be placed in file `$USER.$WAIT_QUEUE.BIG_COMPILE`.

```
/job user_job_name=big_compile output_disposition=wait_queue
job/fortran input=$user.big_job.source binary=$user.big_job.lgo
job/jobend
```

Files placed in the wait queue are saved as the next cycle. Because of this, existing files are not replaced.

The subcatalog `$WAIT_QUEUE` is also used to contain any output files produced by `PRINT_FILE` commands in the job. The name of such a file in the `$WAIT_QUEUE` subcatalog is the name of the output file specified on the `PRINT_FILE` command.

You do not need to create the `$WAIT_QUEUE` subcatalog; the system automatically creates it when necessary.

If your job executes on another system and you have specified `WAIT_QUEUE` on the `OUTPUT_DISPOSITION` parameter, the job's output is returned to the `$WAIT_QUEUE` subcatalog on the system from which you submitted the job.

You can also change the `OUTPUT_DISPOSITION` job attribute to `WAIT_QUEUE` so that all output from your job is placed into the `$WAIT_QUEUE` subcatalog. By doing this, you do not have to specify the `OUTPUT_DISPOSITION` parameter on the `SUBMIT_JOB` or `JOB` command. To set the `OUTPUT_DISPOSITION` job attribute to `WAIT_QUEUE`, enter the following at the system prompt:

```
/change_job_attribute output_disposition=wait_queue
```

Format Effectors

Format effectors are used to control the appearance of data on an output device. Under the right circumstances, the first character of each output line can be interpreted as a format effector. How an output device interprets a format effector depends upon the capabilities of the device. This section discusses how format effectors are used for terminals and printers.

Terminals

There are two types of format effectors for terminals: preprint effectors which control the format before the rest of the line is printed, and postprint effectors which control the format after the line is printed.

The preprint format effectors include:

Character	Action
space	Advance 1 line.
0	Advance 2 lines.
-	Advance 3 lines.
+	Advance 0 lines (overprint).
1	Print at top of screen.
1	Clear screen.
,	Do nothing.

The postprint format effectors include:

Character	Action
------------------	---------------

.	Advance 1 line.
/	Print at current line.

If a file has a `FILE_CONTENT` attribute of `LIST`, `NOS/VE` interprets the first character of each line in the file as a format effector. If column 1 of an output line contains a character that is not a format effector, the system interprets it as the space format effector. Use the `SET_FILE_ATTRIBUTES` command to set the `FILE_CONTENT` attribute to `LIST`.

Every `NOS/VE` command that writes data to or from a file with a `FILE_CONTENT` attribute of `LIST` will interpret format effectors.

Using Format Effectors with `COPY_FILE`

The `COPY_FILE` command interprets format effectors when copying a file with a `FILE_CONTENT` attribute of `LIST` to another file with a `FILE_CONTENT` attribute of `LIST`. If format effectors are not present, `COPY_FILE` assumes the following:

- The first line has a hyphen as a format effector (denoting triple spacing).
- Subsequent lines have a blank space as a format effector (denoting single spacing).

In the following example, file `FORMAT_FILE`, which has a `FILE_CONTENT` attribute of `LIST`, contains the following text:

```
Preprint Test.  
  
1Cleared Screen.  
  Advance 1 line.  
0Advance 2 lines.  
-Advance 3 lines.
```

When you copy file `FORMAT_FILE` to your terminal, the format effectors are interpreted:

```
/copy_file format_file
```

The following portion of the text file appears on the screen:¹

```
Preprint Test.
<OVER>
```

When you press `RETURN`, the screen clears. Then, the remainder of the text file is displayed as follows:

```
Cleared Screen.
Advance 1 line.

Advance 2 lines.

Advance 3 lines.
```

When you assign a local file to your terminal (using the `REQUEST_TERMINAL` command described in Terminal Management later in this manual), its format effectors are, by default, not interpreted. However, you still can have format effectors interpreted by using the `SET_FILE_ATTRIBUTES` command before the file is opened. To do this, enter the following;

```
/request_terminal f=$local.term_f
/set_file_attributes f=$local.term_f fc=list
```

When you are copying to a terminal file, you can suppress format effectors by changing the file's `FILE_CONTENT` attribute to a value other than `LIST`. Once this is changed, `COPY_FILE` will not interpret format effectors when copying to the file.

1. This example shows how text is displayed via a CDCNET connection when the `HOLD_PAGE` and `HOLD_PAGE_OVER` terminal attributes are `TRUE`.

Using Format Effectors with PRINT_FILE

How format effectors are interpreted when using printers depends upon the printer being used. However, the following lists the standard control characters used by NOS/VE and its products:

Character	Action
------------------	---------------

space	Advance 1 line.
0	Advance 2 lines.
-	Advance 3 lines.
+	Advance 0 lines (overprint).
1	Advance to top-of-form.

In addition, if you are using a CDCNET batch station printer and it supports vertical format unit (VFU) load image files to control printing formats, you can use the format effectors defined for standard VFU load files or define additional format effectors by creating a VFU load file. Appendix E lists the format effectors defined by default VFU load files. Refer to the CDCNET Batch Device User Guide for information about creating a VFU file.

Using Standard and Job Files

NOS/VE provides you with a group of standard files which are located in your \$LOCAL catalog. These files serve as the source of input and the destination for output.

You can connect standard files to other files, and any data access request made to the standard file is passed on to the connected files. By default, NOS/VE connects the standard files to a set of job files. The job files are assigned to devices depending on whether the job is interactive or batch. The job files are as follows:

COMMAND

Serves as the main source of input for the SCL interpreter.

INPUT

Serves as the file most programs will read from. For interactive jobs, this file is assigned to the terminal. For batch jobs this file is assigned to the null device class.

OUTPUT

Serves as the file most programs will write to. For interactive jobs, this file is assigned to the terminal. For batch jobs, this file is assigned to the file that is printed by default when the job terminates.

\$JOB_LOG

Represents the job log.

The following table lists the standard files and their initial file connections.

Table 6-1. Standard Files and Initial Connections

File Name	Interactive	Batch	Usage
\$ECHO	None	None	Specifies the file to receive a copy of every SCL command that is interpreted or skipped.
\$ERRORS	OUTPUT	OUTPUT	Specifies the file to which programs and utilities write error information.
\$INPUT	INPUT	INPUT	Specifies the file from which commands and programs read input data.
\$LIST	None	OUTPUT	Specifies the file to which printable output is written.
\$NULL	None	None	Specifies the null device class. Attempts to write to this device class causes data to be discarded as it is written. Attempts to read from this device class always cause an end-of-information status to be returned.
\$OUTPUT	OUTPUT	OUTPUT	Specifies the file to which normal job output is written.
\$RESPONSE	\$JOB_LOG and OUTPUT	\$JOB_LOG	Specifies the file that receives error or informative completion messages from commands. The initial connections of \$RESPONSE cannot be changed.

Creating File Connections

You can create a connection between a subject file and a target using `CREATE_FILE_CONNECTION`. The subject file is usually a standard file, but it can also be a temporary file. In this case, the temporary file must not have been previously created.

When you connect one file to another, any data access request against the subject file is passed to the target file. You can connect a subject file to more than one target file. On input, the access requests are passed only to the most recently connected target file. On output, the access requests to the subject file are passed to each of the target files.

When you connect file `$OUTPUT` to another file, specify the open position of `$EOI` so that write operations will append information to the end of that file.

The following example connects file `$OUTPUT` with file `$USER.OUTPUT_FILE`. The subsequent `DISPLAY_CATALOG` command writes its output to both file `$OUTPUT` (which is connected to file `OUTPUT` by default) and file `OUTPUT_FILE`.

```
/create_file_connection sf=$output f=$user.output_file.$eoi
/disc $user
CATALOG: CATALOG_1
CATALOG: CATALOG_2
  FILE: DATA_FILE_1
  FILE: EPILOG
  FILE: OUTPUT_FILE
  FILE: PROLOG
```


To display the contents of \$USER.OUTPUT_FILE, first delete the current file connection. (If you do not delete the file connection between \$OUTPUT and its target file, a subsequent COPY_FILE command referencing the two files will fail.)

```
/delete_file_connection sf=$output f=output_file
/copy_file i=$user.output_file o=$output
CATALOG: CATALOG_1
CATALOG: CATALOG_2
  FILE: DATA_FILE_1
  FILE: EPILOG
  FILE: OUTPUT_FILE
  FILE: PROLOG
```

The following example connects file \$ECHO to file ECHO_FILE. Notice that the \$EOI open position is used so that ECHO_FILE is not overwritten.

```
/crefc sf=$echo f=echo_file.$eoi
```

As a result of this connection, each subsequent command is echoed to file ECHO_FILE after it is interpreted. Commands written to \$ECHO are preceded by one of the following identifiers that indicates how the command was processed:

- CI Command interpreted.
- CS Command skipped (as in a conditional sequence).

NOTE

Commands executed from procedures residing on files attached in EXECUTE only mode are not echoed.

Displaying File Connections

You can use the `DISPLAY_FILE_CONNECTION` command to display the names of the files currently connected to standard files.

For instance, the following example displays all file connections:

```
/display_file_connection all
:$LOCAL.$ECHO.1 is connected to: :$LOCAL.ECHO_FILE.1.$EOI.
:$LOCAL.$ERRORS.1 is connected to: :$LOCAL.OUTPUT.1.
:$LOCAL.$INPUT.1 is connected to: :$LOCAL.INPUT.1.
:$LOCAL.$LIST.1 is not connected to any files.
:$LOCAL.$OUTPUT.1 is connected to: :$LOCAL.OUTPUT.1.
:$LOCAL.$RESPONSE.1 is connected to: :$LOCAL.RESPONSE_FILE.1,
:$LOCAL.$JOB_LOG.1, :$LOCAL.OUTPUT.1.
```

Deleting File Connections

You can delete a connection between a standard file (and any other connected file) and its target file by using the `DELETE_FILE_CONNECTION` command.

The following example deletes the file connections between `$ECHO` and `ECHO_FILE`:

```
/delete_file_connection sf=$echo f=$local.echo_file
```

You cannot delete the initial file connection between the standard file `$RESPONSE` and files `$JOB_LOG` and `OUTPUT`.

File Connection Considerations

If `NOS/VE` encounters an error when accessing one of the target files, it fulfills your request regarding any remaining target files and reports the error to the user who called the subject file. If `NOS/VE` encounters errors on more than one target file, only the initial error is reported.

When connected files are accessed, each file's attributes are considered separately. Thus, there is no conflict if the page width of a subject file is 80 and the page width of a target file is 132. The subject file and each target file are accessed individually and file attributes of the subject file are irrelevant during access to a target file.

When a file connection is created, the file attributes of the subject file and target file are checked to ensure that they are compatible. A subject file and target file are considered to be compatible when their `FILE_CONTENT` and `FILE_STRUCTURE` attributes are equal (a value of `UNKNOWN` is considered equal to any other value). The files are checked for compatibility each time the files are opened while they are connected.

You can connect a temporary non-standard file to any other file if the temporary non-standard file does not currently exist. However, when you do this and then write data to the temporary file, remember that the data is passed to the target file and no information is actually written to the temporary file. In this situation, when you delete the file connection the temporary file will be empty and assigned to the `NULL` device class.

You can use the `DISPLAY_FILE_ATTRIBUTE` command, the `$FILE` function, or the `$ACCESS_MODE` function displays to display the following:

- Preserved attributes of the target file that was connected first.
- Temporary attributes of the subject file.

When a target file is opened as the result of an access to a subject file, the following file attributes of the subject file apply:

`ACCESS_MODE`

This attribute is always selected. The value is the set of modes common to both the subject file and the target file. Specifically, the access modes for the subject file constitute a subset of the access modes for the target file. For more information, see the description of the `ACCESS_MODES` parameter of the `ATTACH_FILE` command earlier in this chapter.

`OPEN_POSITION`

`NOS/VE` follows a hierarchical set of rules in determining the open position (beginning-of-information, end-of-information, or no positioning) for a target file. These rules are listed in order of precedence. Rules 1, 3, and 4 also assume that the target file is attached via the `ATTACH_FILE` command if the `$ASIS` open position is being used:

1. The open position you specified for the target file when you defined its connection to the subject file.

2. The open position you specified in an `ATTACH_FILE` command of the target file.
3. The open position you specified for the subject file using a file reference.
4. The open position you specified on an `ATTACH_FILE` command of the subject file.

FILE_CONTENT

This attribute is selected only if the target file is new and if no value has been explicitly given for the attribute. The value you assign must conform to one of the following guidelines:

- It must be identical to the `FILE_CONTENT` attribute of the subject file.
- Either file must have a `FILE_CONTENT` value of `UNKNOWN`.
- One file must have a `FILE_CONTENT` value of `ASCII_LOG`, while the other file has a value of `LIST` or `LEGIBLE_DATA`.

If you create a target file as a result of a reference to a subject file, the `FILE_CONTENT` attribute of the target file assumes the same value as the `FILE_CONTENTS` attribute of the subject file.

FILE_STRUCTURE

This attribute is selected only if the target file is new and if no value has been explicitly given for the attribute. The value used is that of the subject file.

RING_ATTRIBUTES

A ring attribute is a value that defines the four ring brackets (read, write, execute, or call) for a target file. This attribute is always selected. If the target file is new, the value used for all three ring attributes is the ring at which the connection was made. If the target file is an old file, the value used for each ring attribute is the greater of the corresponding ring attribute of the target file and the ring at which the connection was made.

Managing Job Logs

A *log* is a record of every given event that occurs while the log is active. One such log is a job log. The *job log* records the following events within your job:

- Command interpretations.
- Conditional statement executions (such as the IF statement).
- Program generated messages.
- System generated messages.
- Recovery of jobs.

NOS/VE creates a job log for every job in the system. The job log becomes active when you begin a job and becomes inactive when you end a job.

Some sites may also support job history logs. A *job history log* records the following events that occur for your user name:

- Submission of jobs to the system.
- Submission of output files to the output queue.

Sites that support job history logging usually deactivate the log only at the end of the day. Because of this, you can interrogate the job history log for events that occurred in previous jobs.

It is possible for you to display the job logs active for your job. You may also write messages to the active job logs for your jobs. NOS/VE provides the following commands for job log management:

CHANGE_MESSAGE_LEVEL

CHANGE_NATURAL_LANGUAGE

DISPLAY_MESSAGE

DISPLAY_LOG

DISPLAY_JOB_HISTORY

DISPLAY_OUTPUT_HISTORY

The following sections describe these commands.

Changing the Message Level

There are two levels of messages displayed: brief and full. By default, the system sets the message level to brief for interactive jobs and to full for batch jobs. To change the level, use the `CHANGE_MESSAGE_LEVEL` command.

For example, to change the message level to full, enter the following command:

```
/change_message_level full
```

When the message level is set to full, error messages contain the severity level, product identifier, and condition code. In addition, any file references appearing in the message are shown with their complete path.

Changing the Message Language

You can specify the language used for messages and help information produced within your job. However, message and help modules must be available for the language you specify (if the specified modules are not available, NOS/VE uses the `US_ENGLISH` modules). For more information, see the NOS/VE Object Code Management manual.

To specify the language for messages, use the `CHANGE_NATURAL_LANGUAGE` command. The following example changes the natural language to Spanish:

```
/change_natural_language natural_language=spanish
```

Putting Messages into a Job Log

You can put messages into your active job log and job history log, and the display message area of a job by using the `DISPLAY_MESSAGE` command:

- If you put a message into a job log, you can see the message using the `DISPLAY_LOG` command.
- If you put a message into a job history log, you can display the message using the `DISPLAY_JOB_HISTORY` and the `DISPLAY_OUTPUT_HISTORY` commands.
- If you put a message in the job's display message area, you can display the message using the `DISPLAY_JOB_STATUS` command.

Putting Messages into an Active Job Log

The following example puts a message in the job log and then displays the previous two lines of the log:

```
/dism message='Message for job log.' to=job
/disl 2
10:08:43.089.CI.dism 'Message for job log.' to=job
10:08:43.107.PR.Message for job log.
10:08:47.219.CI.disl 2
```

Putting Messages into a Job History Log

The following example puts a message in the job history log:

```
/dism message='Message for history log.' to=history
```

To display the job history log, use the `DISPLAY_JOB_HISTORY` command. For an example of using history log messages, refer to the `SHOW_JOB_HISTORY` example in the online Examples manual.

Putting Messages into the Display Message Area

The following procedure uses the `DISPLAY_MESSAGE` command to issue a display message every 100 iterations. The job can then use the `DISPLAY_JOB_STATUS` command to monitor its progress.

```
proc proc1
for i=1 to 10000
if $mod(i,100)=0 then
st=$strrep(i)
dism message='i is '//st to=job_message
ifend
forend
procend
```

While procedure PROC1 is executing, the user occasionally checks the procedure's status using the network keystroke sequence, %s (see chapter 3 for more information on network keystroke sequences).

```

/proc1
%s
CPU_Time_Used           :   Job Mode- 7.288
                        :   Monitor Mode- 1.660

Display_Message        :   i is 200
Page_Faults           :   Assigned- 1610
                        :   From Disk- 1255
                        :   Reclaimed- 672

%s
CPU_Time_Used           :   Job Mode- 12.958
                        :   Monitor Mode- 1.699

Display_Message        :   i is 1400
Page_Faults           :   Assigned- 1618
                        :   From Disk- 1255
                        :   Reclaimed- 689

```

You can use this technique in a batch job you submitted in order to monitor its progress from other jobs. Use the DISPLAY_JOB_STATUS command to display the status of the job.

Displaying the Current Job Log

To display the contents of an active job log, use the DISPLAY_LOG command. Each log entry is in the following format:

```
hh:mm:ss.mmm.mo.text
```

The time the log entry was made is shown as the hour (hh), minute (mm), second (ss), and millisecond (mmm). The message origin (mo) is one of the following:

- CI Command interpreted.
- CS Command skipped (as in a conditional sequence).
- PR Program-generated message.
- RC Job recovery.
- SY System message.

The following example displays the last 16 lines of an active job log:

```

/display_log display_options=16
09:21:44.066.CI.change_message_level brief
09:21:58.187.CI.set_file_attributes data_file
09:22:09.325.CI.copy_file data_file
09:22:09.379.PR. --ERROR-- FSP$OPEN_FILE was issued for
    file, DATA_FILE, which does not exist.
09:22:21.311.CI.scu
09:22:24.597.CI.quit
09:22:24.775.PR. Task complete SCP$SCU
09:22:24.775.PR.      job time =      0.072
    monitor time =      0.018 page faults = 59
    max working set = 110
09:22:28.765.CI.cham1 full
09:22:37.351.CI.copy_file data_file
09:22:37.372.PR. --ERROR AM 1016-- FSP$OPEN_FILE was
    issued for file, :NVE.SARETT.DATA_FILE,
09:22:37.372.PR. which does not exist.
09:22:43.843.CI.scu
09:22:52.917.CI.quit
09:22:52.974.PR. Task complete SCP$SCU
09:22:52.975.PR.      job time =      0.075
    monitor time =      0.016 page faults = 52
    max working set = 110
09:23:38.809.CI.display_log display_options=16

```

Each of the entries in the preceding example was created when a command was read either from an interactive file or the main command file (\$LOCAL.COMMAND) of a batch job, or each time a task finished executing. Note that some utilities suppress the logging of subcommands. The information written to the job log at task completion indicates the task name, the time in seconds it took the task to execute, page fault data, and the maximum working set size of the data while it was executing. For more information on this data, see the NOS/VE Object Code Management manual.

Displaying a Job History Log

To display the history of a job, use the `DISPLAY_JOB_HISTORY` command. (Job history logging must be activated at your site.)

For example,

```
/display_job_history
$0990_0102_AAD_1367 JOB_QUEUEING_STARTED          02/06/88  14:51:43
                  UJN=SARETT, LOGIN_USER=:NVE.SARETT,
                  CONTROL_USER=:NVE.SARETT, STATION=AUTOMATIC

$0990_0102_AAD_1367 JOB_INITIATED                02/06/88  14:51:43

$0990_0102_AAD_1367 PRINT_PLOT_FILE_EXECUTED    02/06/88  15:01:02
                  SFN=$0990_0102_AAD_1413, OUTPUT_DESTINATION=NVE,
                  OUTPUT_DESTINATION_USAGE=DUAL_STATE,
                  USER_FILE_NAME=PROLOG

$0990_0102_AAD_1367 SUBMIT_JOB_EXECUTED         02/06/88  15:19:42
                  SJN=$0990_0102_AAD_1457, JOB_DESTINATION=NVE,
                  JOB_DESTINATION_USAGE=VE, UJN=SARETT

/
```

Displaying Output History

To display the history of a job's output, use the `DISPLAY_OUTPUT_HISTORY` command. (Job history logging must be activated at your site.)

For example,

```
/display_output_history
$0990_0102_AAD_1367 PRINT_PLOT_FILE_EXECUTED    07/31/87  15:21:02
                  SFN=$0990_0102_AAD_1413, OUTPUT_DESTINATION=NVE,
                  OUTPUT_DESTINATION_USAGE=DUAL_STATE,
                  USER_FILE_NAME=PROLOG

$0990_0102_AAD_1367 OUTPUT_QUEUEING_STARTED    07/31/87  15:21:03
                  LOGIN_USER=:NVE.SARETT, SFN=$0990_0102_AAD_1413,
                  CONTROL_USER=:NVE.SARETT, STATION=AUTOMATIC

$0990_0102_AAD_1367 PRINT_PLOT_INITIATED       07/31/87  15:21:03
                  SFN=$0990_0102_AAD_1413

$0990_0102_AAD_1367 PRINT_PLOT_TERMINATED     07/31/87  15:21:15
                  SFN=$0990_0102_AAD_1413

/
```

Displaying Job Resource Limits

When you are validated on the system, your site administrator sets up system resource limits for your user name. The jobs you execute under your user name are limited to the resources allowed for your user name. It is possible for you to further restrict the resources used by a job running under your user name. With the `CHANGE_JOB_LIMIT` command, you can change the job limits for the following resources:

- Central processing (CP) time
- System resource units (SRUs)
- Number of tasks permitted in the job

To display a job's resource limits, use the `DISPLAY_JOB_LIMITS` command. A display of your job limits similar to the following example appears:

```
/display_job_limits
```

Limit Name	Accumulator	Resource Limit	Abort Limit
-----	-----	-----	-----
CP_TIME	3	126663740	140737488
SRU	1	UNLIMITED	UNLIMITED
TASK	1	10	11

For each of the limit names, the system also displays values for the following:

- | | |
|-----------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Abort Limit | Maximum number resources allowed for your user name. If your job reaches the abort limit, your job is logged out. |
| Accumulator | Total number of resources used by your job. |
| Resource Limit | Maximum number of resources allowed before your job causes a resource condition. You can change the resource limits for a particular job to any value between the current value of the accumulator and the job abort limit. |

Setting Multiprocessing Options

Some computer systems contain two central processing units (CPUs). For systems with dual CPUs, you can specify the CPU on which you want your job to run, or you can specify that you want your job to run on both processors.

You can find out how many processors your system has by using the `TOTAL_PROCESSORS` keyword on the `$MAINFRAME` function.

To set these multiprocessing options, use the `SET_MULTIPROCESSING_OPTIONS` command. The following examples show how to use the `SET_MULTIPROCESSING_OPTIONS` command:

```
/set_multiprocessing_options      Job runs on either P0 or P1
                                   (whichever was the previous
                                   default), but not both.

/set_multiprocessing_options ..    Job runs on P0 and/or P1.
./mp=on

/set_multiprocessing_options ..    Job runs on P0 only.
./mp=off
```

Setting Job Sense Switches

A *sense switch* is a section of memory that an executing program can interrogate as a boolean value. By turning a sense switch on or off, you can externally control the execution of a program. For information on interrogating sense switches from within a program, see your programming language's language definition manual.

The `SET_SENSE_SWITCH` command turns a job's software managed sense switches on or off. Each job has eight switches associated with it that are identified by integer values 1 through 8. A system operator can set sense switches for any job. You can set sense switches only for jobs you own.

For example:

```
/set_sense_switch on=(1,2,3)
/set_sense_switches job_name=$0855_0104_pdq_0861 on=1 off=2
/set_sense_switches job_name=$0855_0104_pdq_0862 ..
../off=(1,2,3,4,5,6,7,8)
```

Note that you can interrogate sense switches from within an SCL procedure using the `SWITCHn` keyword of the `$JOB` function. You can also display their current settings using the `DISPLAY_JOB_ATTRIBUTES` command.

Creating and Using Variables	7-2
Variable Names	7-3
Creating Variables	7-4
Creating Variables Implicitly	7-4
Creating Variables Explicitly	7-5
Array Variables	7-6
Assigning Initial Values	7-7
Values Assigned Implicitly	7-7
Values Assigned Explicitly	7-9
Scope of Variables in a Block Structure	7-10
Statically Linked Blocks	7-10
Specifying the Scope of a Variable	7-12
Example: Job Block Structure	7-13
Examples of Variables	7-16
Implicit Variable Creation	7-16
Explicit Variable Creation	7-17
Deleting Variables (DELETE_VARIABLE)	7-19
Displaying a List of Variables (DISPLAY_VARIABLE_LIST)	7-19
Variable References	7-20
Referencing Array Variables	7-20
Referencing Status Variables	7-20
SCL Types	7-22
Integer Type	7-23
Specifying a Number Base	7-23
Specifying Hexadecimal Integers	7-24
Specifying a Sign	7-25
Using Spaces	7-25
Examples of Valid and Invalid Integers	7-26
Real Type	7-27
Specifying a Real Number	7-27
Examples of Valid and Invalid Real Numbers	7-28
String Type	7-29
Boolean Type	7-31
Status Type	7-32
File Type	7-34
Name Type	7-35
Key Type	7-35
Any Type	7-36
Application Value Name	7-36

Expressions 7-37
Integer Expressions 7-38
String Expressions 7-39
Logical Expressions 7-40
Relational Expressions 7-41
Combinations of Expressions 7-44
Displaying the Value of an Expression 7-46

This chapter presents discussions of the following topics in SCL:

- Variables, including how to create and delete, use and display variables.
- Types, including acceptable types for parameters and variables and the rules governing each type.
- Expressions, including rules for syntax and operators.

Examples are provided with each discussion.

Creating and Using Variables

A *variable* is a name that refers to a data item whose value can be changed. SCL supports the following types of variables:

Integer

Real

String

Boolean

Status

The environment in which a variable is accessible is called the *scope* of the variable. Unless otherwise specified, a variable is assigned a local scope, that is, its use is restricted to the block in which it is created.

Variables can reside in the following kinds of blocks:

- *Job block.* The job block exists once a job begins and includes all statements from the LOGIN command to the LOGOUT command, including any prolog or epilog commands.
- *Utility block.* A utility block is established by a program that processes utility subcommands; when the utility terminates, the block is no longer defined.
- *Procedure block.* A procedure block is created when an SCL procedure is called; when the procedure terminates, the block is no longer defined. A procedure can be called anywhere within a job but is considered a separate block at the point at which it is called. Thus, any local variables created within a procedure are known only to the procedure.
- *WHEN block.* A WHEN block is created when a condition occurs for which a WHEN statement has been defined. For more information about the WHEN statement, see chapter 8, Commands Streams and Condition Processing.

Variables created in other blocks reside in the enclosing block from those listed above. In the following example, the variable COUNTER is created within a FOR/FOREND block, but it resides in the enclosing (interactive) job block.

```
/FOR counter=1 TO 10 DO
:
FOR/FOREND
/
```

A variable exists from the time it is defined until it is explicitly deleted or until the block in which it resides terminates.

Variable Names

Variable names follow the same rules and conventions as other SCL names with the following additional restrictions:

- The boolean constant names, TRUE, FALSE, YES, NO, ON and OFF cannot be used for variable names.
- A variable name cannot begin with the dollar sign (\$) character.
- A variable name cannot be the same as that of another variable in the same block.
- A variable name in a procedure should not be the same as a parameter name in the same procedure.

Creating Variables

You can create SCL variables either *implicitly* by assigning a value to a variable name, or *explicitly* using the `CREATE_VARIABLE` command.

Creating Variables Implicitly

You can create variables implicitly in one of the following ways:

- By using an undefined variable name as the object of an assignment statement. For example:

```
/example_2 = 5
```

`EXAMPLE_2` is an integer variable whose value is 5.

If you implicitly create a variable by equating it to an existing variable, the new variable inherits its attributes from the variable to which it is equated. For example:

```
new_example = example_2
```

Like `EXAMPLE_2`, `NEW_EXAMPLE` is an integer variable whose value is 5.

- By using an undefined variable name as the control variable of a `FOR` statement. The value you assign must be an integer. For example:

```
FOR i=1 to 10 DO
  .
  . (statement list)
  .
FOREND
```

For more information about the `FOR/FOREND` statement, see chapter 8, `Command Streams and Condition Processing`.

Creating Variables Explicitly

You can create variables explicitly by using the `CREATE_VARIABLE` command. The following is a list of parameters and parameter requirements for the `CREATE_VARIABLE` command:

names, name, n	: list of name = \$required
kind, k	: list 1..2 of any = integer
dimension, d	: range of integer -2147483647..2147483647 = 1
value, v	: any = \$optional
scope, s	: name = local
status	: var of status = \$optional

The following example creates a simple integer variable whose initial value is 3:

```
/create_variable n=example_1 k=integer v=3
```

NOTE

You cannot create a variable of type real using the `CREATE_VARIABLE` command. You can, however, create variables of type real implicitly.

Array Variables

An *array* is a set of values of a particular data type identified by a single variable name. For example, a boolean array named ANSWERS could contain all the true or false answers to a set of survey questions.

The values in an array variable (also called elements) each occupy a specific position in the array. Each of these positions is represented by an integer within the range of -2147483647 to 2147483647.

The position occupied by the first element in an array is called the *lower bound*. The position occupied by the last element in an array is called the *upper bound*.

When you create an array variable with the CREATE_VARIABLE command, you specify the upper and lower bounds of the array using the DIMENSION parameter. For example, the following command creates an array whose first element is referenced by the integer 1 and whose last element is referenced by the integer 10:

```
/create_variable n=digits k=integer d=1..10
```

If you omit the DIMENSION parameter, the variable is assumed not to be an array, that is, it has dimensions of 1..1.

Assigning Initial Values

You can assign a value to a variable either when you create it, or at a later time. A variable must have a value, however, before it can be read. The value you assign to a variable when you create it depends in part on whether you create the variable implicitly or explicitly.

Values Assigned Implicitly

When you implicitly create a variable, the following rules apply:

- The new variable inherits the attributes of the variable or expression to which it is equated. In the following example, the variable `NEW_VARIABLE` is a string variable whose value is the string *Doe, John*.

```
new_variable = 'Doe, John'
```

In the following example, the variable `NAME` has all of the attributes of variable `NEW_VARIABLE`; it is a string variable whose value is the string *Doe, John*.

```
name = new_variable
```

- If you implicitly create an array variable, you must equate the new variable to an existing array. The new array has the same type, values, and upper and lower bounds as the array to which it is equated. For example:

```
new_array = old_array
```

Subscripts are not allowed when you implicitly create an array variable. For example, the following statements are not valid:

```
a_new_array(10) = 0
```

```
b_new_array(4) = old_array
```

- If you implicitly create a status variable using an assignment statement, you must equate the new variable to an existing status variable or use the \$STATUS function. The new status variable is identical to the status variable to which it is equated. For example:

```
new_status = old_status
```

Field references are not allowed when you implicitly create a status variable. For example, the following statement is not valid:

```
new_status.normal = true
```

You can achieve the equivalent of the previous statement, however, using the \$STATUS function. The \$STATUS function has parameters which correspond to the fields of a status variable, and can be used to assign values to those fields. For example:

```
new_status = $status(true)
```

The above statement assigns the boolean value TRUE to the NORMAL field of the variable NEW_STATUS.

The fields of a status variable are discussed later in this chapter.

Values Assigned Explicitly

When you create a variable using the `CREATE_VARIABLE` command, you can optionally assign the variable an initial value using the `VALUE` parameter. The following restrictions apply:

- You cannot use the `VALUE` parameter to assign values to specific elements in an array. Each element in an array is initialized with the value you specify.

Once an array variable has been created, you can assign values to specific array elements using assignment statements. For example:

```
/create_variable n=array_1 k=integer d=10
/array_1(8) = 80
```

- You cannot use the `VALUE` parameter to assign values to the fields of a status variable. You must either use the system default (see below) or the `$STATUS` function. For example:

```
/create_variable n=new_status k=status v=$status(false,'xx',0)
/array_1(8) = 80
```

- If you do not specify an initial value, the following default values are assigned:

Type	Default Initial Value
Boolean	FALSE
Integer	0
Status	NORMAL field = TRUE
String	" (null string)

Variables of types other than those listed above are not given default values.

Scope of Variables in a Block Structure

The scope of a variable defines the blocks in which the variable can be accessed. Variables are by default local to the block in which they are created. A local variable is accessible to the following blocks:

- The block in which the variable is defined.
- Blocks which are statically linked to the block in which the variable is defined.

Statically Linked Blocks

Two blocks are statically linked when one block is defined directly within another. A statically linked block has access to local variables residing in the enclosing block.

The following blocks are statically linked to the blocks in which they are called:

- Utility blocks
- Blocks created by the `INCLUDE_FILE` command
- Blocks created by the `INCLUDE_LINE` command
- Blocks delimited by the following control statements:

<code>BLOCK/BLOCKEND</code>	<code>REPEAT/UNTIL</code>
<code>FOR/FOREND</code>	<code>WHEN/WHENEND</code>
<code>IF/IFEND</code>	<code>WHILE/WHILEND</code>
<code>LOOP/LOOPEND</code>	

For those statically linked blocks in which variables can reside (utilities and WHEN/WHENEND) the following considerations apply:

- If a variable is created with the same name and type as a locally accessible variable, references to that variable name default to the variable residing in the current block. In the following example, the string variable VAR_1 is created in the job block, and referenced in the nested utility block. Another string variable named VAR_1 is then created and referenced in the utility block.

```

/create_variable var_1 k=string v='Job Block Variable'
/create_object_library
COL/display_value var_1
Job Block Variable
COL/create_variable var_1 k=string v='Utility Block Variable'
COL/display_value var_1
Utility Block Variable
COL/

```

- When the variable residing in the utility block is deleted or the block terminates, references to VAR_1 refer to the variable residing in the job block:

```

COL/delete_variable var_1
COL/display_variable var_1
Job Block Variable
COL/

```

Specifying the Scope of a Variable

If you create a variable using the `CREATE_VARIABLE` command, you can use the `SCOPE` parameter to specify the scope of the variable.

You can specify any of the following values for the `SCOPE` parameter:

LOCAL

The variable is local to the current block. A variable with this scope can be accessed only by the block in which it resides, and by blocks to which it is statically linked.

XDCL

The variable is externally declared (XDCL). A variable with this scope can be referenced by other blocks if a variable with identical `KIND` and `DIMENSION` parameters is created with a scope of `XREF` in those blocks.

XREF

The variable is externally referenced (XREF). A variable with this scope can reference another block if a variable with identical `KIND` and `DIMENSION` parameters is created with a scope of `XDCL` in that block.

JOB

Causes the variables to be created in the job block with an `XDCL` scope. If the current block is not the job block, an `XREF` declaration in the current block is made. A variable with a scope of `JOB` is implicitly accessible from `SCL` commands at the same level, from within a utility environment, from within a block, and from within another task. However, a `JOB` scope does not allow a variable to be implicitly accessed from within an `SCL` procedure. To access a variable with a `JOB` scope from an `SCL` procedure, you must create the variable within the procedure with an `XREF` scope.

name

Causes the variables to be created in the utility block specified by `name` with an `XDCL` scope. If the current block is not the utility block, an `XREF` declaration in the current block is made.

Example: Job Block Structure

Figure 7-1 illustrates a job block. The following considerations apply to this illustration:

- The job block encompasses everything from LOGIN to LOGOUT.
- Each inner block is a procedure block created by calling a procedure.
- The job block contains two subordinate blocks created by the execution of procedures A and D.
- Variables created in the job block and outside of blocks A and D with an XDCL scope are accessible within the job block. Other blocks can access these variables if they create a variable with the same name and type with a scope of XREF.
- Local variables created within block A are known only to block A; local variables created within block D are known only to block D.
- Once blocks A and D cease to exist (that is, once the procedure terminates), local variables within their scope also cease to exist.
- Blocks A and D each contain one subordinate procedure block. Block B is subordinate to block A, and block E is subordinate to block D.
- Blocks B and E each contain one subordinate procedure block. Block C is subordinate to block B, and block F is subordinate to block E.
- The rules for the scope of variables in blocks B, C, E, and F are the same as those described for blocks A and D.

Table 7-1 illustrates the scope of variables shown in figure 7-1. Each variable is assumed to have been created with the default scope of LOCAL.

Table 7-1. Scope of Variables

Variable	Defined in Block	Accessible to Block
i	JOB	JOB, A, B, C, D, E, F
a	A	A, B, C
b	B	B, C
c	C	C
j	JOB	JOB, D, E, F
d	D	D, E, F
e	E	E, F
f	F	F
k	JOB	JOB
l	A	A
m	B	B
n	D	D
o	E	E

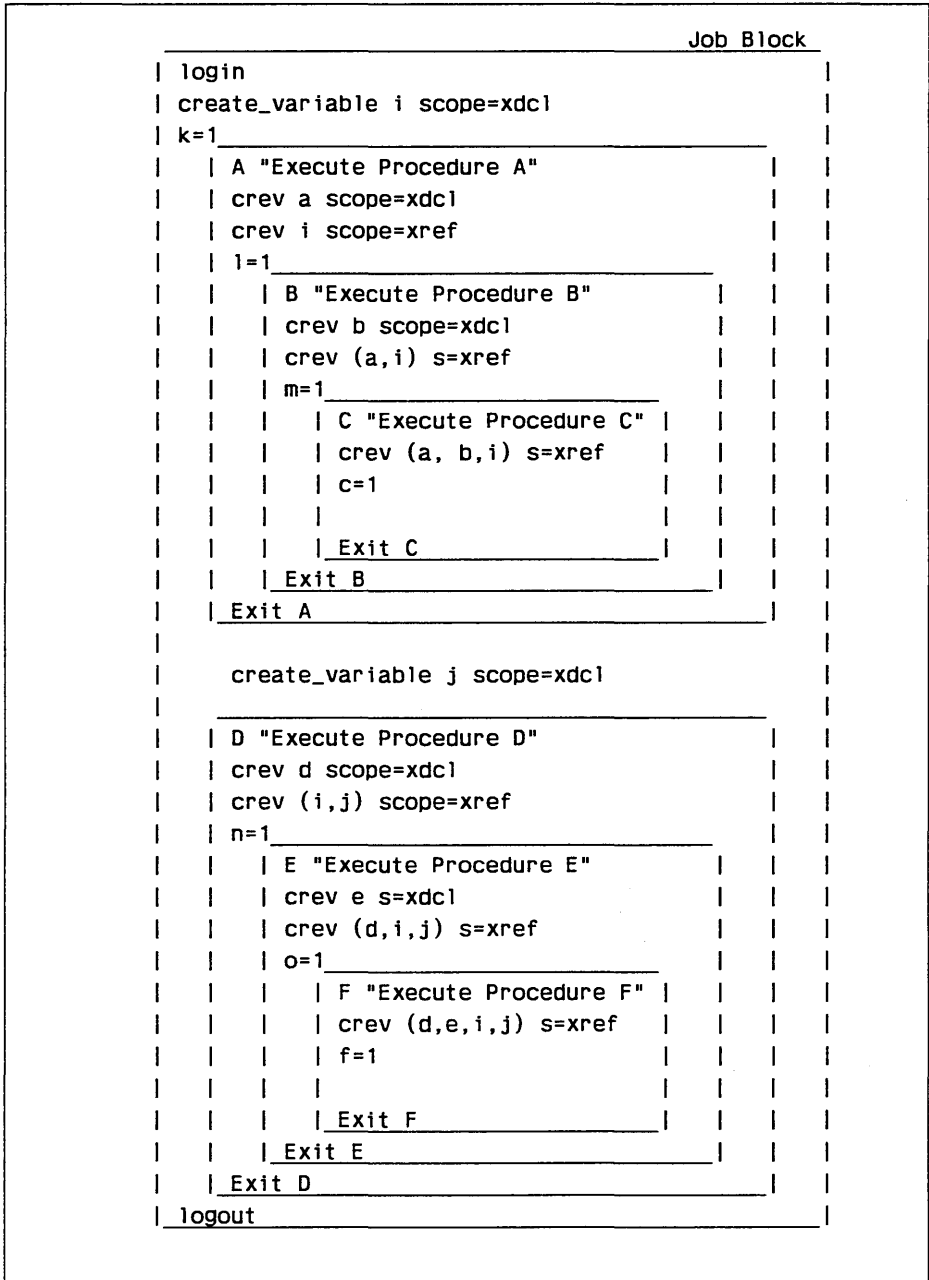


Figure 7-1. Job Block Structure

Examples of Variables

This section presents examples of implicitly and explicitly created variables.

Implicit Variable Creation

If a status variable by the name of `JOB_STATUS` currently exists, the following statement creates a status variable named `NEW_JOB_STATUS` that is identical to the variable `JOB_STATUS`:

```
new_job_status = job_status
```

The next example creates an array variable named `LIST` with 10 elements, and then creates a variable named `NEW_LIST`, which has the same type and values as `LIST`:

```
create_variable n=list d=10 k=string  
new_list = list
```

Since `LIST(1)` is a string variable, `NEW_LIST(1)` is also a string variable.

The next example creates a status variable named `NEW_STATUS`, and then creates a status variable named `OLD_STATUS`, which contains all of the fields of `NEW_STATUS`:

```
create_variable n=new_status k=status  
old_status = new_status
```

The following statement creates a boolean variable named `FAILED` with the value contained in the `NORMAL` field of the status variable `OLD_STATUS`:

```
failed = old_status.normal
```

Explicit Variable Creation

This section presents examples of variables created with the `CREATE_VARIABLE` command. The last example briefly illustrates the scope of variables in SCL procedures.

The following example creates a variable named `GLOBAL_STATUS`, of type `STATUS`, with a scope of `JOB`:

```
/create_variable n=global_status k=status s=job
```

The `GLOBAL_STATUS` variable can be referenced externally from any block (via the `SCOPE=XREF` parameter).

The next example creates a variable named `DONE` of type `boolean`. Its initial value is set to `FALSE`, and it is given a scope of `XDCL`. Other blocks can reference the `DONE` variable if they create the same variable with a scope of `XREF`.

```
/create_variable n=done k=boolean v=false s=xdcl
```

The next example creates a variable named `SHORT_STRING` of type `string`. Its initial value is set to `'US'`, with a maximum length of 2 characters. Since the `SCOPE` parameter is omitted, `LOCAL` scope is assumed.

```
/create_variable n=short_string v='US' k=(string,2)
```

The next example creates an integer array (`ARRAY_1`) of 10 elements that can be referenced as 1 through 10:

```
/create_variable n=array_1 k=integer d=1..10
```

The next example creates a string array (`ARRAY_2`) of 10 elements that can be referenced as -5 through 4:

```
/create_variable n=array_2 k=string d=-5..4
```

In the next example, the `VALUE` parameter sets a boolean variable `FLAG` to `TRUE`:

```
/create_variable n=flag k=boolean v=true
```


The following example shows how to use a variable with a scope of JOB to pass values in and out of a procedure. The examples are based on the following procedure (stored in file \$LOCAL.MY_PROC):

```
PROC my_proc ()  
  
    create_variable n=job_string k=string s=xref  
    job_string = 'new string value'  
    display_value '--MY_PROC completed--'  
  
PROCEND my_proc
```

Entering the following command creates variable JOB_STRING with a scope of JOB:

```
/create_variable n=job_string v='string value' s=job
```

The following series of statements illustrates how the value of JOB_STRING is changed within procedure MY_PROC:

```
/display_value job_string  
string value  
/my_proc  
--MY_PROC completed--  
/display_value job_string  
new string value  
/
```

The first instance of the DISPLAY_VALUE command displays the initial value of variable JOB_STRING. The procedure MY_PROC is then executed. The second DISPLAY_VALUE command displays the value of variable JOB_STRING after it has been changed within the procedure.

Deleting Variables (DELETE_VARIABLE)

You can delete a variable in either of the following ways:

- *Explicitly*, using the DELETE_VARIABLE command either from the block in which the variable is defined or from a block to which the defining block is statically linked.
- *Implicitly*, when the block in which the variable is defined ceases to exist.

The following example uses the DELETE_VARIABLE command to remove variables named COUNT and LOOPS:

```
/delete_variable (count,loops)
```

Displaying a List of Variables (DISPLAY_VARIABLE_LIST)

To display a list of the variables accessible from the current block, use the DISPLAY_VARIABLE_LIST command. The system displays the variable names starting with the most recently created variable.

The following is a sample display of variables created in the job block. VARIABLE_1 was created first, and VARIABLE_4 was created last.

```
/display_variable_list
LOCAL VARIABLES IN JOB

variable_4          variable_3
variable_2          variable_1
osv$status
```

OSV\$STATUS is a status variable created by the system in the job block at the beginning of the job. The OSV\$STATUS variable is discussed in chapter 8, SCL Command Streams and Condition Processing.

Variable References

You refer to a variable by the name you assigned it at the time of its creation. Array variables and variables of type status each have additional considerations, and are discussed in the following sections.

Referencing Array Variables

Each element in an array occupies a particular position between the upper and lower bounds of the array. This position is represented by an integer subscript. To reference an element of an array, specify the subscript in the following format:

```
variable_name(subscript)
```

The subscript can be any integer expression. For example:

```
next_entry_location(last+1)
```

The following considerations also apply to array variables:

- A space is not allowed between the variable name and its subscript.
- If you omit the subscript from an array variable reference, the reference is to the entire variable.

Referencing Status Variables

A status variable has three fields in which information about the completion status of a command is stored. The status variable and its fields are described later in this chapter. To reference a specific field of a status variable, use the following format:

```
variable_name.field name
```

Field name can be any of the following keywords:

```
NORMAL  
CONDITION  
TEXT
```

The following example references the `CONDITION` field of a status variable named `LOCAL_STATUS`:

```
local_status.condition
```

The following considerations apply to status variables:

- Spaces are not allowed surrounding the period between the variable name and its field designation.
- If you omit the field designation from a status variable reference, the reference is to the entire variable.

The following are valid variable references:

Variable Reference	Description
<code>data_list_1(10)</code>	References the 10th element of variable array <code>DATA_LIST_1</code> .
<code>stat.condition</code>	References the <code>CONDITION</code> field of status variable <code>STAT</code> .
<code>stat_array(4).normal</code>	References the <code>NORMAL</code> field of the fourth element in the array of status variables named <code>STAT_ARRAY</code> .

The following are invalid variable references:

Invalid Reference	Reason
<code>stat.next</code>	Invalid field specification (<code>NEXT</code>) for the status variable.
<code>list (4)</code>	Space between the variable name and the subscript is not allowed.
<code>stat . normal</code>	Spaces surrounding the period are not allowed in the field specification.

SCL Types

In SCL, variables and parameters have a specific data type, such as integer or boolean, associated with them (parameters may have more than one data type associated with them). Consequently, when you create an SCL variable or define a parameter for a procedure, you define a particular type for it. Likewise, when you assign a value to an existing variable or specify a value for a parameter, the value you specify must match the data type defined for the variable or parameter.

SCL supports the following data types for parameters:

- Integer
- Real
- String
- Boolean
- Status
- File
- Name
- Key
- Any
- Application Value Name

Currently, SCL supports the following subset of types for variables:

- Integer
- Real
- String
- Boolean
- Status

This section defines each of the types listed and discusses the syntax requirements for each.

Defining variables is discussed in the previous section. Defining procedure parameters is discussed in chapter 9, *Writing SCL Procedures and Command Utilities*.

Integer Type

An integer is a value representing one of the numbers 0, +1, -1, +2, -2, and so on. SCL supports integer values within the range from -9,223,372,036,854,775,808 to +9,223,372,036,854,775,807. The minimum and maximum integer values are returned by the \$MIN_INTEGER and \$MAX_INTEGER functions respectively.

Specifying a Number Base

SCL supports integer values in bases two through sixteen. To specify an integer in a base other than decimal, you must include a trailing radix. If you omit the radix, decimal is assumed.

When you enter a trailing radix, surround it with opening and closing parentheses, as shown in the following examples:

Integer(Radix)	Decimal Equivalent
1011010(2)	90
14(8)	12
93(10)	93
1F(16)	31

Specifying Hexadecimal Integers

To specify an hexadecimal (base sixteen) integer use a combination of numeric characters and the following letters:

A B C D E F
a b c d e f

SCL makes no distinction between uppercase and lowercase letters in hexadecimal integer values.

You must begin an hexadecimal integer with a digit. When you specify a hexadecimal integer that begins with an alphabetic character, you must add a leading zero. This is necessary to avoid the ambiguity that arises with certain representations. For example, the 16th element of an array whose name is A1 is indicated as follows:

A1(16)

This is also the hexadecimal representation of the decimal value 161. To avoid this ambiguity, you must enter the hexadecimal representation as follows:

0A1(16)

Specifying a Sign

You can optionally precede an integer with a sign (ASCII plus or minus character). If you omit the sign, a plus is assumed. For example:

+100	A positive number
-1AF02(16)	A negative number
700(8)	A positive number

Using Spaces

The following is a set of guidelines for using spaces when entering integer values:

- Precede and follow an integer with either a delimiter, such as a comma or space, or an operator, such as * (for multiplication).
- Do not enter a space between the digits and the radix.
- You can enter spaces after the opening parenthesis and before the closing parenthesis of the radix.
- Avoid using spaces between a number and its sign, especially when using real numbers in expressions for parameter values.

Examples of Valid and Invalid Integers

The following are valid integer values:

```

123456789    +704264(8)
0FFFF(16)    -63( 10)
-101(2)      10101010101( 2 )

```

The following are invalid integer values:

Invalid Integer	Reason
100200(2)	Contains a digit, 2, that is invalid in a binary representation
FFF(16)	Does not begin with a digit, although it is an hexadecimal value.
345(17)	Contains an invalid radix specification (17).
0fff	Does not contain a radix indicating an hexadecimal value.
123 (8)	Contains a space between the digits and the radix.

Real Type

A real number consists of the following elements:

- A *mantissa*, which is a sequence of digits containing a decimal point (period).
- A Trailing exponent (optional) with an optional sign.

Specifying a Real Number

Specify a real number as follows:

- Include at least one digit on each side of the decimal point.
- Use decimal numbers only.
- Separate the mantissa from the exponent with either the letter E or D (in uppercase or lowercase).
- Delimit a real number at both ends (with spaces, for example).
- Do not enter a space in the mantissa, in the exponent, or between the mantissa and the exponent.
- Avoid using spaces between the sign and number digits, especially when using real numbers in expressions for parameter values.
- Create real numbers only in the range from 4.8E-1234 to 3.2E+1232.

The real number value is the mantissa multiplied by 10 raised to the power specified by the exponent. SCL maintains real numbers in normalized double-precision, floating-point format.

NOTE

Currently, the following additional considerations apply to real numbers:

- You can use real numbers as values for both implicitly created variables and parameters.
 - You cannot specify REAL as the type for an explicitly created variable.
 - Real number arithmetic operations are not supported.
 - Real numbers with leading signs are not supported.
-

Examples of Valid and Invalid Real Numbers

The following are valid real numbers:

12.26E2 34.0e5
3.0d-3 0.003

The following are invalid real numbers:

Invalid Real Value	Reason
.33e-5	Does not have at least one digit on each side of the decimal point.
5.6(8)	Includes a number base.
-3.2 d-4	Contains an embedded space and a leading sign.

Real Type

A real number consists of the following elements:

- A *mantissa*, which is a sequence of digits containing a decimal point (period).
- A Trailing exponent (optional) with an optional sign.

Specifying a Real Number

Specify a real number as follows:

- Include at least one digit on each side of the decimal point.
- Use decimal numbers only.
- Separate the mantissa from the exponent with either the letter E or D (in uppercase or lowercase).
- Delimit a real number at both ends (with spaces, for example).
- Do not enter a space in the mantissa, in the exponent, or between the mantissa and the exponent.
- Avoid using spaces between the sign and number digits, especially when using real numbers in expressions for parameter values.
- Create real numbers only in the range from 4.8E-1234 to 3.2E+1232.

The real number value is the mantissa multiplied by 10 raised to the power specified by the exponent. SCL maintains real numbers in normalized double-precision, floating-point format.

NOTE

Currently, the following additional considerations apply to real numbers:

- You can use real numbers as values for both implicitly created variables and parameters.
 - You cannot specify REAL as the type for an explicitly created variable.
 - Real number arithmetic operations are not supported.
 - Real numbers with leading signs are not supported.
-

Examples of Valid and Invalid Real Numbers

The following are valid real numbers:

12.26E2 34.0e5
3.0d-3 0.003

The following are invalid real numbers:

Invalid Real Value	Reason
.33e-5	Does not have at least one digit on each side of the decimal point.
5.6(8)	Includes a number base.
-3.2 d-4	Contains an embedded space and a leading sign.

String Type

A string is any sequence of ASCII characters surrounded by apostrophes. The maximum string length is 256 characters. The following is an example of a string:

```
'This is a string.'
```

You can enter a string containing a single space, as follows:

```
' '
```

To denote an empty string, enter two consecutive apostrophes, as in the following example:

```
''
```

In a string, the apostrophes serve as delimiters and are not part of the string itself. To add an apostrophe to the string, represent it as two consecutive apostrophes as follows:

```
'You can''t use a single apostrophe within a string.'
```

The following considerations apply to string values:

- If you create a string variable implicitly, or if you do not specify the length of a string when explicitly creating a string variable, the system assigns the variable a default maximum length of 256 characters.
- At any given time, a string variable has a current length equal to the length of the string last assigned to it.
- If a string is assigned a value that is shorter than its maximum length, the remaining characters become blanks.
- If a string is assigned a value that is longer than its maximum length, excess characters are truncated and the string's current length is its maximum length.

The following are valid string values:

Valid String	Explanation
'A. B. Smith'	Ordinary string.
''''	String of one apostrophe.
'This long .. string is .. continued'	Continued string.

The following are invalid string values:

Invalid String	Reason
'This string is not terminated.	The final apostrophe is missing.
'You can't have a single .. apostrophe.'	Within a string, a single apostrophe must be represented by two consecutive apostrophes.
'This long string is not correctly continued onto a second line.'	The ellipsis (..) is missing at the end of the lines that are continued (after <i>not</i> and after <i>second</i>).

Boolean Type

A boolean value is a value that is either TRUE or FALSE.

You can represent a boolean value of TRUE with any of the following names (uppercase or lowercase):

```
TRUE
YES
ON
```

You can represent a boolean value of FALSE with any of the following names (uppercase or lowercase):

```
FALSE
NO
OFF
```

The following are valid boolean assignments:

```
done = false
debug_mode = ON
print_listing_reply = no
```

The following are invalid boolean assignments:

Invalid Boolean

Assignment	Reason
debug_mode = .. correct	Improper specification of TRUE value.
done = .f.	Improper specification of FALSE value.

Status Type

A status value is a record that contains the completion status of a command. Each status record consists of three fields. These fields are described in the following table.

Table 7-2. Status Variable Fields

Field Name	Field Kind	Description
NORMAL	Boolean	Specifies whether the command completed without an error (TRUE) or with an error (FALSE).
CONDITION	Integer	If the command had an error, this integer specifies the condition code of the diagnostic message for the aborted command. This field also contains a 2-character product identifier.
TEXT	String of length 256	This string contains message parameters that are substituted into the basic diagnostic message corresponding to the condition encountered.

If the command completes normally, the NORMAL field is set to TRUE and all other fields are undefined. If the command cannot be completed, the NORMAL field is set to FALSE; the other fields return the conditions and the diagnostic message parameters describing the error that occurred.

The following considerations apply to status values:

- Spaces are not allowed surrounding the period between the variable name and its field designation.
- If you omit the field designation from a status variable reference, the reference is to the entire variable.

The following are valid status references:

Valid Status Value	Reason
<code>stat.condition</code>	References the CONDITION field of status variable STAT .
<code>stat_array(4).normal</code>	References the NORMAL field of the fourth element in the array of status variables named STAT_ARRAY .

The following are invalid status references:

Invalid Status Value	Reason
<code>stat.next</code>	Invalid field specification (NEXT) for the status variable.
<code>stat . normal</code>	Spaces surrounding the period are not allowed in the field specification.

File Type

NOTE

The **FILE** type is currently supported for parameter values only and may not be used when defining variables.

A value of type **FILE** can be a reference either to a catalog or to a file. Whether the reference is to a file or a catalog depends on the use to which the value is put. For example, the following command results in an error:

```
/display_catalog c=$local.$errors
```

Although the **CATALOG** parameter on the **DISPLAY_CATALOG** command accepts a value of type **FILE**, the command itself requires that the value supplied be a catalog reference. Because your **\$LOCAL** catalog cannot have subcatalogs, the value in the above example refers to a file called **\$ERRORS** in the **\$LOCAL** catalog.

The format for file and catalog references is as follows:

```
:family.username.catalog(s).filename.cycle.position
```

Each element of a file reference must be a valid SCL name. The entire reference cannot exceed 256 characters including colon (:) and period (.) separators. For more information about referencing files and catalogs, see chapter 4, Catalog and File Management.

The following are valid file references:

Valid File

References	Reason
<code>\$user.my_data.2</code>	References the second cycle of file MY_DATA located in the \$USER catalog.
<code>.smith.output.\$eoi</code>	References the END_OF_INFORMATION position for file OUTPUT in SMITH'S master catalog.

The following are invalid file references:

Invalid File

Invalid File Reference	Reason
<code>\$local.output.test_1</code>	<code>\$LOCAL</code> catalog cannot have subcatalogs (OUTPUT).
<code>\$user . notes</code>	Spaces surrounding separators are not allowed in file or catalog references.

Name Type

NOTE

The NAME type is currently supported for parameter values only and may not be used when defining variables.

A value of type NAME is any valid SCL name, that is, any combination of alphanumeric and special characters as long as the name is less than 31 characters and does not begin with a number.

For more information about SCL names, see chapter 2, The System Command Language.

For more information about defining parameters of type NAME, see chapter 9, Writing SCL Procedures and Command Utilities.

Key Type

NOTE

The KEY type is currently supported for parameter values only and may not be used when defining variables.

A value of type KEY (also called a *keyword*) is generally one of several options provided for a given command. For example, the `CHANGE_INTERACTION_STYLE` command allows you to specify one of two keyword options for the `STYLE` parameter: `LINE` or `SCREEN`.

For more information about defining parameters of type KEY, see chapter 9, Writing SCL Procedures and Command Utilities.

Any Type

NOTE

The ANY type is currently supported for parameter values only and may not be used when defining variables.

Specifying type ANY (also called the UNION type) provides for the case where any one type is applicable. For example, the VALUE parameter of the DISPLAY_VALUE command allows you to specify an expression of any type. Values specified for parameters of type ANY must meet the restrictions and syntax requirements for the particular type specified.

Application Value Name

NOTE

Application value names are currently supported for parameter values only and may not be used when defining variables.

When writing SCL procedures, you can define your own value type by specifying a name of your own choosing when defining a parameter's type. This user-defined type is called an application value.

When you specify an application value, you can also specify the procedure which is called to evaluate the parameter value. If no procedure is specified, the parameter value is returned as a string with no evaluation performed.

For more information about application values, see the Cybil System Interface manual.

Expressions

An expression is the representation of one or more operations performed to compute a value. A single value or variable appearing alone also constitutes an expression. This chapter focuses on expressions in which constants and variables are combined by operators.

NOTE

Real number arithmetic operations are not supported.

Expressions are used to perform the following kinds of computation:

- Integer addition
- Integer subtraction
- Integer multiplication
- Integer division
- Integer exponentiation
- String concatenation
- Logical difference and sum
- Logical complement
- Logical product
- Relational comparisons

In an expression, the system evaluates operands within parentheses first, with the innermost parentheses taking precedence.

When the system encounters an SCL-defined constant or function in an expression, it evaluates the constant or function (and any supplied arguments) and uses the resulting value in the expression.

An expression can contain any number of data items, but after it is evaluated, it represents a single value.

In an expression used in an assignment or control statement, you can surround the operator with spaces; in an expression used in a parameter, you cannot enter spaces surrounding the operators.

Integer Expressions

An integer expression is one that is evaluated as an integer value.

In integer expressions, you can use the following operators:

Operator	Use
+	Indicates that two integer operands are to be added.
-	Indicates that the right operand is to be subtracted from the left operand.
*	Indicates that two integer operands are to be multiplied.
/	Indicates that the left operand is to be divided by the right operand. Standard integer division is employed: the result is the integer quotient with any remainder discarded.
**	Indicates that the integer preceding the operator is exponentiated by the integer following the operator.

Except when using an integer expression as a parameter value, you can precede and/or follow the operator with spaces.

The following series of assignment statements (each one affecting the next) is an example of using integer expressions.

Statement	Meaning
<code>i = 1</code>	Assigns the integer 1 to the variable I.
<code>j = i+2</code>	Assigns the sum of I and 2 to the variable J. Therefore, J is equal to 3.
<code>k = 2**j</code>	Assigns the result of 2 raised to the Jth power to the variable K. Therefore, K is equal to 8.
<code>k = k/3</code>	Assigns the quotient of the variable K divided by 3 to the variable K. Therefore, K is equal to 2.

String Expressions

A string expression is one that is evaluated as a string.

In string expressions, you can use the following concatenation operator:

```
//
```

This operator joins the two string operands to form a single string. Except when using an expression as a parameter value, you may precede and/or follow the operator with spaces.

Included in the following assignment statements is an example of the concatenation operator.

Statement	Meaning
<code>string_1 = 'First part'</code>	Assigns the string 'First part' to the string variable <code>STRING_1</code> .
<code>string_2 = ' last part'</code>	Assigns the string ' last part' to the string variable <code>STRING_2</code> .
<code>s = string_1//string_2</code>	Concatenates the values of string variables <code>STRING_1</code> and <code>STRING_2</code> and places the result in string variable <code>S</code> . The value of <code>S</code> is as follows: 'First part last part'

Logical Expressions

A logical expression is one that is evaluated as either true or false. In a logical expression, all operands must be boolean values. The result of a logical expression is always a boolean value.

In logical expressions, you can use the following operators:

Operator	Use
OR	Performs a logical sum (inclusive OR) of two operands. If one operand is TRUE, the expression is evaluated as TRUE; otherwise, it is evaluated as FALSE. If the left operand is TRUE, the right operand is not evaluated.
XOR	Performs a logical difference (exclusive OR) between two operands. If one operand is TRUE, but not both, the expression is evaluated as TRUE; otherwise, it is evaluated as FALSE.
AND	Performs a logical product of two operands. If both operands are TRUE, the expression is evaluated as TRUE; otherwise, it is evaluated as FALSE. If the left operand is FALSE, the right operand is not evaluated.
NOT	Performs a logical complement of an operand. If the operand is TRUE, the expression is evaluated as FALSE. If the operand is FALSE, the expression is evaluated as TRUE.

You must precede and follow a logical operator with one or more spaces. The following logical expression is valid:

```
(string='a') or (string='b')
```

The following logical expression is invalid:

```
(string='a')or(string='b')
```

The following assignment statements are examples of using logical expressions:

Statement	Meaning
<code>y = true; z = false</code>	Creates boolean variables Y and Z.
<code>x = y OR z</code>	If either Y or Z is TRUE, X is TRUE. Since Y is TRUE, X is TRUE.
<code>x = NOT y</code>	X is the complement of Y. Since Y is TRUE, X is FALSE.
<code>x = y AND z</code>	X is TRUE only if both Y and Z are TRUE. Since Z is FALSE, X is FALSE.
<code>x = y XOR z</code>	X is TRUE because Y is TRUE and Z is FALSE (one operand is TRUE but not both).

Relational Expressions

A relational expression is one that expresses a relationship between values of the same kind (for example, two string operands or two integer operands). The result of a relational expression is always a boolean value.

In relational expressions, you can use the following operators:

Operator	Result
<code>></code>	When the operand to the left of this operator is greater than the operand to the right, the expression is evaluated as TRUE; otherwise, it is evaluated as FALSE.
<code><</code>	When the operand to the left of this operator is less than the operand to the right, the expression is evaluated as TRUE; otherwise, it is evaluated as FALSE.
<code>>=</code>	When the operand to the left of this operator is greater than or equal to the operand to the right, the expression is evaluated as TRUE; otherwise, it is evaluated as FALSE.

Operator	Result
< =	When the operand to the left of this operator is less than or equal to the operand to the right, the expression is evaluated as TRUE; otherwise, it is evaluated as FALSE.
=	When the operand to the left of this operator is equal to the operand to the right, the expression is evaluated as TRUE; otherwise, it is evaluated as FALSE.
< >	When the operand to the left of this operator is not equal to the operand to the right, the expression is evaluated as TRUE; otherwise, it is evaluated as FALSE.

Except when using a relational expression as a parameter value, you can precede and/or follow the operator with spaces.

The following additional rules apply to relational expressions:

- FALSE is evaluated as less than TRUE.
- String comparisons are made according to the ASCII collating sequence. All ASCII characters are collated according to their ASCII codes. For example, the # character is ASCII code 23 (hexadecimal) and so is higher in the collating sequence than the \$ character, which is ASCII code 24 (hexadecimal). A listing of the ASCII character set is in appendix C. Uppercase and lowercase characters are not equivalent in the ASCII collating sequence, and therefore are not equivalent in string comparisons.
- When the system compares two strings of unequal length, it treats the shorter string as though space characters extended it to the right to the length of the larger string.

The following series of statements illustrate how to use relational expressions:

Statement	Meaning
<pre> i = 5 j = 1 s = 'abc ' t = 'abcde' y = true z = false </pre>	<p>Initializes integer variables I and J, string variables S and T, and boolean variables Y and Z.</p>
<code>i > j</code>	I (5) is greater than J (1). Therefore, the expression is evaluated as TRUE.
<code>i < j</code>	I (5) is not less than J (1). Therefore, the expression is evaluated as FALSE.
<code>s = t</code>	Before the comparison, string S is extended on the right with spaces so that its length equals the length of string T. S ('abc ' followed by an additional space) is not equal to T ('abcde'). Therefore, the expression is evaluated as FALSE.
<code>y <> z</code>	Y (TRUE) is not equal to Z (FALSE). Therefore, the expression is evaluated as TRUE.
<code>i >= j</code>	I (5) is greater than J (1). Therefore, the expression is evaluated as TRUE.
<code>s <= t</code>	The first 3 characters of both S ('abc ') and T ('abcde') are equal (abc). However, for S, character 4 is a space, whereas for T character 4 is d. The ASCII code for a space is 20 (hexadecimal), whereas the ASCII code for d is 64 (hexadecimal). Therefore, S is less than T, and the expression is evaluated as TRUE.

Combinations of Expressions

You can combine only certain kinds of operands with a particular operator. The valid combinations and the kind of the result are shown in table 7-3.

When the precedence of operations is not specified with parentheses, the system performs the operations in the order shown in the table. For example, ****** is evaluated before ***** in the following expression:

$a*b**c$

Thus the preceding expression is identical to:

$a*(b**c)$

When the order of precedence is equivalent, evaluation is performed from left to right. For example, the result of the following expression is 4:

$8 / 4 * 2$

If you reversed the preceding expression's elements, the result would be 1.

$2 * 4 / 8$

Table 7-3. Operand and Operator Combinations

Left Operand	Operator	Right Operand	Result	Order of Precedence¹
Integer	**	Integer	Integer	1
Integer	* or /	Integer	Integer	2
Integer	+ or -	Integer	Integer	3
None	+ or -	Integer	Integer	3
String	//	String	String	4
Integer	Relational	Integer	Boolean	5
String	Relational	String	Boolean	5
Boolean	Relational	Boolean	Boolean	5
None	NOT	Boolean	Boolean	6
Boolean	AND	Boolean	Boolean	7
Boolean	OR or XOR	Boolean	Boolean	8

1. When order of precedence is equivalent, operators are evaluated from left to right.

Displaying the Value of an Expression

To display the value of an expression, use the `DISPLAY_VALUE` command. The following is a list of parameters and parameter requirements for the `DISPLAY_VALUE` command:

`value, values, v` : list of any = \$required
`output, o` : file = \$output
`status` : var of status = \$optional

If you specify more than one expression for the `VALUE` parameter, the system displays each value on a separate line.

The following examples demonstrate use of the command with expressions of several different types.

```
/display_value 2**7-4  
124
```

```
/display_value ('Line 1','Line 2','Line 3')  
Line 1  
Line 2  
Line 3
```

```
/display_value ($max_integer,$min_integer)  
9,223,372,036,854,775,807  
-9,223,372,036,854,775,808
```

```
/create_variable name=stat kind=status  
/display_value $variable(stat,kind)  
STATUS
```

In addition, you can use the `DISPLAY_VALUE` command to determine hexadecimal, octal, or decimal equivalents. In the three examples that follow, the first displays the decimal equivalent of an hexadecimal number, the second displays the hexadecimal equivalent of a decimal number, and the third displays the octal equivalent of a decimal number.

```
/display_value 1*0FFFFFF(16)  
16777215
```

```
/display_value 1(16)*16777215  
0FFFFFF(16)
```

```
/display_value 1(8)*16777215  
7777777(8)
```

For a more formal way of converting integers, see the description of the `$STRREP` function in the `NOS/VE Commands and Functions` manual.

SCL Command Streams and Condition Processing

Block Structure	8-2
Structuring a Command Stream	8-3
Grouping Statements into a Block (BLOCK/BLOCKEND)	8-5
Causing Unlimited Repetition of a Statement List (LLOOP/LOOPEND).	8-6
Causing Preconditional Repetition of a Statement List (WHILE/WHILEND).	8-7
Causing Postconditional Repetition of a Statement List (REPEAT/UNTIL)	8-8
Causing Controlled Repetition of a Statement List (FOR/FOREND)	8-9
Controlling the Flow of a Command Stream	8-10
Causing the Next Iteration of a Repetitive Statement (CYCLE)	8-10
Exiting from a Structured Statement (EXIT)	8-12
Exiting from a Procedure or Utility (EXIT)	8-14
Executing Statement Lists Conditionally (IF/IFEND)	8-15
Condition Processing	8-16
Error Processing	8-17
Error Processing Example	8-18
Testing for Specific Error Conditions	8-19
Condition Handling	8-20
Establishing a Condition Handler (WHEN/WHENEND)	8-22
Exiting from a WHEN Block (CONTINUE)	8-24
Canceling a Condition (CANCEL)	8-25
Suspending Command Processing (WAIT)	8-26
Changing System Environments	8-26
Establishing a New Environment (PUSH)	8-27
Restoring the Previous Environment (POP)	8-28

SCL provides commands and control statements that structure and control the command stream by organizing and processing it in blocks.

This chapter describes the SCL block structure and the statements you can use to perform the following operations:

- Structuring a command stream into blocks.
- Controlling the flow of a command stream.
- Executing statement lists conditionally.
- Error processing using status variables.
- Condition handling.
- Suspending command processing.
- Changing the system environment.

Block Structure

In SCL a *block* is a physical or logical grouping of data. A NOS/VE job is typically organized as several SCL blocks. At a minimum, each job has one block called the *job block*. The job block contains the SCL statements from job initiation to job termination (including prolog and epilog processing initiated by these commands).

The following is a list of entities in SCL which constitute their own block:

- Job
- Utility
- Procedure
- The following control statements:

BLOCK/BLOCKEND	REPEAT/UNTIL
FOR/FOREND	WHEN/WHENEND
IF/IFEND	WHILE/WHILEND
LOOP/LOOPEND	

- The INCLUDE_FILE command
- The INCLUDE_LINE command

NOTE

The INCLUDE_FILE and INCLUDE_LINE commands are described in chapter 9, Writing SCL Procedures and Command Utilities, along with the following related topics:

- Reading lines from a file (ACCEPT_LINE)
 - Writing lines to a file (PUT_LINE)
 - Inserting files into the command stream (INCLUDE_FILE)
 - Inserting lines into the command stream (INCLUDE_LINE)
 - Inserting commands into the command stream (INCLUDE_COMMAND)
-

The following considerations apply to blocks:

- Within a command stream, a block of statements is treated as a separate entity until a specified terminating condition or statement is encountered.
- Blocks created entirely within another block are subordinate to the outer block and are said to be *statically linked* to that block. For example, the block created by a call to a utility is statically linked to the block which issued the call. A statically linked block has access to the variables defined in the outer block. For more information, see chapter 7, SCL Variables, Types, and Expressions.

Structuring a Command Stream

SCL provides control statements that enable you to organize a command stream into blocks of statements which can then be processed in a sequential, conditional, or repetitive manner.

Some control statements have a clause which initiates the block and a clause which terminates the block. This type of control statement is called a *structured* statement.

Each structured statement has the following format:

```
label: structured statement begin clause
      statement list
structured statement end clause label
```

The optional label is any valid SCL name; it is a convenient way of referring to a structured statement. Flow control statements (described later in this chapter) can transfer control within or out of a structured statement by referencing the statement label.

A label precedes the structured statement it identifies and is separated from the structured statement by a colon delimiter. Except for a REPEAT statement, a label can optionally follow a structured statement end clause.

The following example illustrates a labelled LOOP statement:

```
multiple_input: LOOP
  :
  "statement list for multiple input"
  :
LOOPEND multiple_input
```

The following conditions apply to the format of labelled statements:

- The beginning and ending labels must be identical.
- A label cannot be referenced by procedures called from within a structured statement or by statement lists introduced by `INCLUDE_FILE` or `INCLUDE_LINE` commands (described in chapter 9).
- No space can precede the colon.
- A space following the colon is optional.
- The scope of a statement label is restricted to the statement it labels. That is, it is not possible to refer to a label on a structured statement from outside that statement.

The following structured statements are described in this section:

- `BLOCK/BLOCKEND`
- `LOOP/LOOPEND`
- `WHILE/WHILEND`
- `REPEAT/UNTIL`
- `FOR/FOREND`

Grouping Statements into a Block (BLOCK/BLOCKEND)

To group a sequence of statements into a block, use the BLOCK/BLOCKEND statement. Exiting from the block occurs either when the last statement in the statement list is executed or through explicit transfer of control via an EXIT statement. Flow control statements are described later in this chapter.

The format of the BLOCK/BLOCKEND statement is as follows:

```
label: BLOCK
      statement list
BLOCKEND label
```

The BLOCK/BLOCKEND statement is used primarily to designate a group of statements which can be exited if a particular condition exists.

The following example creates a block named MAIN. The statement list includes an EXIT flow control statement that causes an exit from block MAIN when the tested condition is TRUE. If the tested condition is FALSE, the EXIT statement is inhibited.

```
main: BLOCK
      :
      exit when not status.normal
      :
BLOCKEND main
```


Causing Unlimited Repetition of a Statement List (LOOP/LOOPEND)

To cause a statement list to be repeated an unlimited number of times, use the LOOP/LOOPEND statement. This statement is useful, for example, when you want to read an undetermined number of lines from a file.

NOTE

Exiting from a LOOP statement is possible only via an EXIT statement.

The format of the LOOP/LOOPEND statement is as follows:

```
label: LOOP
      statement list
LOOPEND label
```

The following example uses a LOOP statement to read lines from file INPUT. Each line read is stored in string variable INPUT_STRING. When a null input line is encountered, the execution of the loop is ended; otherwise the string variable is written to file \$OUTPUT.

```
input_string = ''
read_input: LOOP
  accept_line input_string input
  EXIT read_input WHEN $strlen(input_string) = 0
  display_value input_string
LOOPEND read_input
```

When this loop is executed, the following interaction takes place:

```
SUPPLY INPUT_STRING testing
testing
SUPPLY INPUT_STRING
/
```

Pressing RETURN after the SUPPLY INPUT_STRING prompt results in the variable INPUT_STRING having a length of 0. This causes the EXIT statement condition to be TRUE. The loop is then exited.

Causing Preconditional Repetition of a Statement List (WHILE/WHILEND)

To perform conditional repetition of a statement list, use the WHILE/WHILEND statement. Prior to each iteration of the statement list, the boolean expression in the WHILE statement is evaluated. If the expression is TRUE, the statement list is executed; if the expression is FALSE, control passes to the statement following the WHILEND clause.

The format of the WHILE/WHILEND statement is as follows:

```
label: WHILE boolean expression DO
    statement list
WHILEND label
```

The following example computes the factorial of a variable named FACTORIAL_OF:

```
/factorial_of = 5    "Compute the factorial of 5."
/last_value = 1     "Value for first loop."
/factorial: while factorial_of > 1 do
while/last_value = factorial_of * last_value
while/factorial_of = factorial_of - 1
while/whilend factorial
/display_value last_value
120
/
```

Causing Postconditional Repetition of a Statement List (REPEAT/UNTIL)

To perform conditional repetition of a statement list, use the REPEAT/UNTIL statement. After each iteration of the statement list, the boolean expression within the UNTIL clause is evaluated. If this expression is evaluated as FALSE, the statement list is executed again. If the expression is evaluated as TRUE, control passes to the statement following the UNTIL clause. Thus, a REPEAT statement is always executed at least once.

The format of the REPEAT/UNTIL statement is as follows:

```
label: REPEAT
      statement list
UNTIL boolean expression
```

The following example reads lines from file INPUT until a null input line is entered:

```
/line = ''
/repeat
repeat/accept_line v=line i=input
repeat/display_value line
repeat/until line = ''
SUPPLY LINE Line 1
Line 1
SUPPLY LINE

/
```

Pressing return after the SUPPLY LINE prompt signals end of input for the loop.

Causing Controlled Repetition of a Statement List (FOR/FOREND)

To control repetition of a statement list, use the FOR/FOREND statement.

The format of the FOR/FOREND statement is as follows:

```
label: FOR variable = initial TO final BY step DO
  statement list
FOREND label
```

The following example displays each character in a string on a separate line:

```
/string = 'down hill'
/blanks = ' '
/for i = 1 to $strlen(string) do
for/display_value $substr(blanks,1,i)//..
for../$substr(string,i,1)
for/forend
d
 o
  w
   n

    h
     i
      l
       l
```

Controlling the Flow of a Command Stream

SCL provides control statements that control the flow of a structured statement. These *flow control statements* are as follows:

- The CYCLE statement causes the next iteration of a repetitive statement.
- The EXIT statement causes control to pass to one of the following:
 - The statement immediately following a structured statement.
 - The caller of an SCL procedure or utility.

Causing the Next Iteration of a Repetitive Statement (CYCLE)

Use the CYCLE statement to begin the next iteration of a repetitive statement. If any of the conditions specified in the CYCLE statement occur, the current iteration stops at the point where the condition occurs, and the next iteration or cycle begins.

The format of the CYCLE statement is as follows:

```
CYCLE label  
      WHEN boolean expression
```

The CYCLE statement can be used within the statement list of the following structured statements:

- LOOP statement: The occurrence of a CYCLE statement causes control to be transferred to the first statement in the statement list of the LOOP statement.
- WHILE statement: The boolean expression that conditionally repeats the statement is evaluated, and the WHILE statement terminates or continues from that point.
- REPEAT statement: The boolean expression that conditionally repeats the statement is evaluated, and the REPEAT statement terminates or continues from that point.
- FOR statement: The FOR statement terminates with the final iteration or continues with the next iteration.

Figure 8-1 illustrates how the CYCLE statement functions. The following considerations apply to this illustration:

- It assumes there are conditional statements that would cause control to go to loop 2 preceding the CYCLE statement.
- If the first CYCLE is executed, the LOOP statement labelled LOOP1 is cycled.
- If the second CYCLE is executed, the LOOP statement labeled LOOP1 is also cycled. Thus, both LOOP2 and LOOP1 are effectively cycled.
- If the third CYCLE statement is executed, the LOOP statement labelled LOOP2 is cycled and control remains in LOOP2.

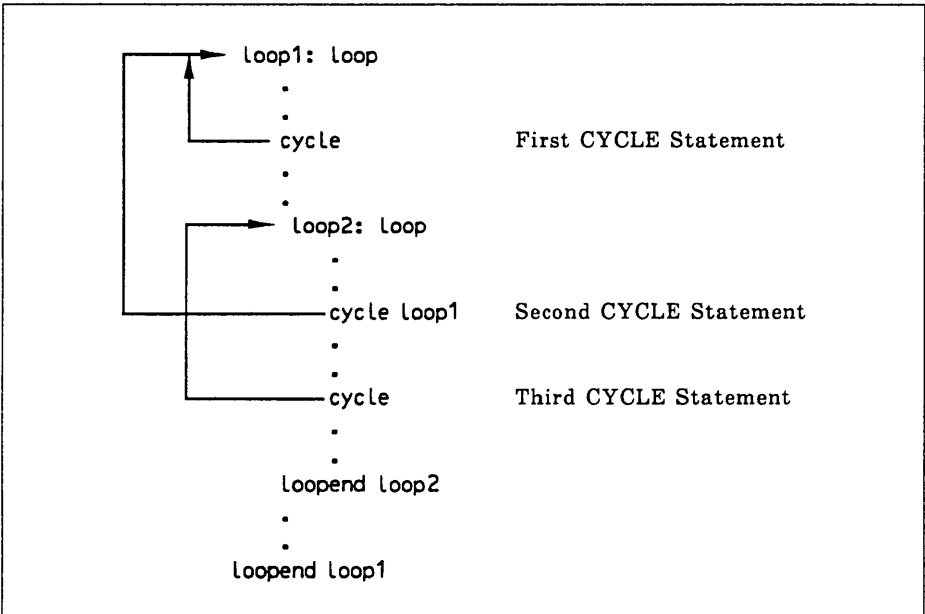


Figure 8-1. CYCLE Statement Use

Exiting from a Structured Statement (EXIT)

To cause your job to transfer control out of a structured statement or any block, use the EXIT statement. Execution continues with the statement following the structured statement.

The format of the EXIT statement is as follows:

```
EXIT designator
    WHEN boolean expression
    WITH status expression
```

The following considerations apply to the EXIT statement

- The term *designator* applies to the label of a structured statement, the name of an active procedure, the name of an active utility, or the keywords PROCEDURE and UTILITY.
- The *status expression* used with the WHEN clause must be a procedure's status value.
- You can use the WHEN and WITH clauses either separately or together. If you use them together, you can list either clause first.

Figure 8-2 illustrates how the EXIT statement functions. The following considerations apply to this illustration:

- It assumes that, preceding the EXIT statement, there are conditional statements that would cause control to go to BLOCK2.
- If the first EXIT is executed, control is transferred to the next statement after the LOOPEND statement labeled BLOCK1, and BLOCK1 is terminated.
- If the second EXIT is executed, control is also transferred to the next statement after the LOOPEND statement labeled BLOCK1. Thus, both BLOCK2 and BLOCK1 are terminated.
- If the third EXIT statement is executed, control is transferred to the next statement after the LOOPEND statement labeled BLOCK2, and BLOCK1 remains active.

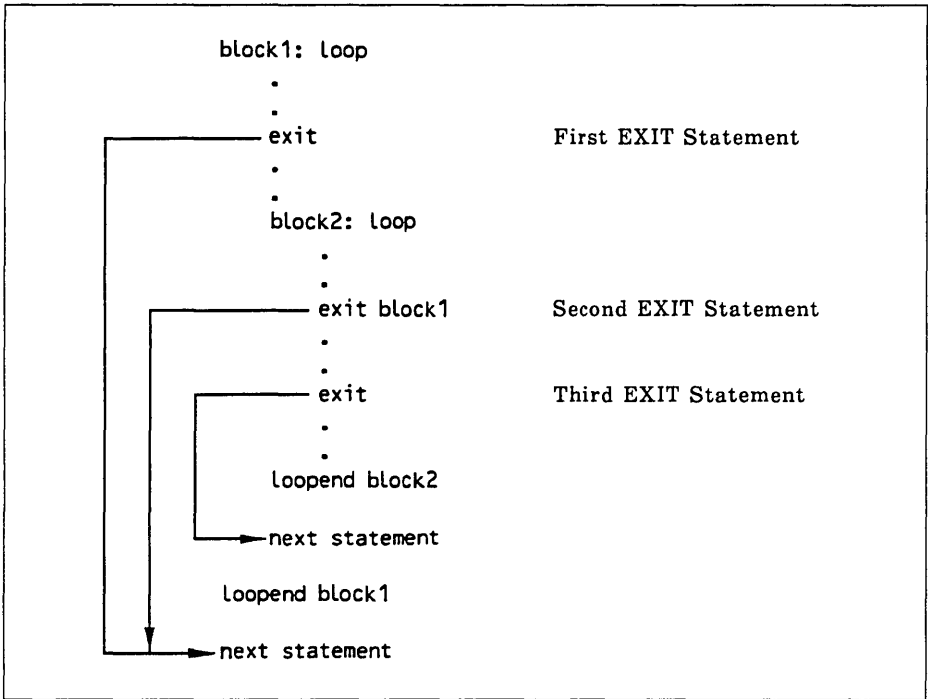


Figure 8-2. EXIT Statement Use

In the following example, the FOR statement is exited when the system encounters an abnormal status:

```

process: FOR i = 1 TO 10 DO
  :
  EXIT process WHEN NOT status.normal
  :
FOREND process

```


Exiting from a Procedure or Utility (EXIT)

Exiting from a Procedure or Utility (EXIT)

To return control to the caller of an SCL procedure or utility, use the EXIT statement. Execution resumes at the statement following the procedure or utility call.

The following example exits a procedure with an abnormal status:

```
PROC example (status)
  :
  EXIT example WITH ..
  $status(false,'US',1,'Proc Example Failed')
  :
PROCEND example
```

If the EXIT statement is executed, the following output is returned:

```
/example
--ERROR-- CC=US 1 TEXT=?Proc Example Failed
```

Executing Statement Lists Conditionally (IF/IFEND)

To conditionally execute one of several statement lists, use the IF/IFEND statement in conjunction with ELSEIF and ELSE clauses. Statement execution is based on the evaluation of associated boolean expressions.

The format of the IF/IFEND statement is as follows:

```

IF boolean expression THEN
    Statement list
ELSEIF boolean expression THEN
    Statement list
ELSE
    Statement list
IFEND

```

The following is an example of an IF statement within an SCL procedure (refer to the NOS/VE Commands and Functions manual for descriptions of the \$PARAMETER and \$VALUE functions). Assume that an integer parameter named INT is passed to the procedure. The following IF statement causes the procedure to report whether the integer is negative, zero, or positive:

```

PROC pos_or_neg (int : integer)
  IF $value(int) = 0 THEN
    display_value $parameter(int)//' is Zero'
  ELSEIF $value(int) > 0 THEN
    display_value $parameter(int)//' is Positive'
  ELSE
    display_value $parameter(int)//' is Negative'
  IFEND
PROCEND pos_or_neg

```

Condition Processing

SCL provides the following mechanisms for specifying the action to be taken when an abnormal condition occurs:

- *Error processing*

This mechanism is available when the STATUS parameter is included in a command and the command terminates with an abnormal status.¹

- *Condition handling*

This mechanism can be implemented when an abnormal condition occurs for a command under the following circumstances:

- The STATUS parameter is not specified for the command.
- A syntax error is encountered for the command.
- The job resource limit has been reached.
- A pause break has been entered.

Error processing and condition handling are discussed in the sections that follow.

1. For an example of how to process errors in statements that do not include the STATUS parameter, see the description of the INCLUDE_LINE command in chapter 9, Writing SCL Procedures and Command Utilities.

Error Processing

All NOS/VE commands have an optional parameter called STATUS. When you specify the STATUS parameter, you must supply a previously declared variable of type STATUS as its value. This variable is used by the SCL interpreter to hold the completion status of the command.

If you include the STATUS parameter on a command, the SCL interpreter proceeds to the next command even if an abnormal condition is encountered. Most commands do not inform you of an error if you include the STATUS parameter. However, succeeding commands can check the contents of the status variable and alter the flow of statements, based upon the occurrence of abnormal conditions.

If you do not include a value for the STATUS parameter and an error occurs, the SCL interpreter skips succeeding commands in the current input block.

The following blocks are considered input blocks:

- Job block
- Utility block
- Procedure block
- WHEN/WHENEND block
- INCLUDE_FILE block
- INCLUDE_LINE block

For more information about variables of type STATUS, see chapter 7, SCL Variables, Types, and Expressions.

Error Processing Example

The following example creates a status variable; calls a command; and, depending upon the outcome of the command, conditionally executes a set of statements:

```
create_variable name=local_status kind=status
attach_file file=$user.data_library status=local_status
IF NOT local_status.normal THEN
  :
  "Perform error processing."
  :
IFEND
```

In the example, the following activity takes place:

1. The `CREATE_VARIABLE` command creates a status variable named `LOCAL_STATUS`.
2. An `ATTACH_FILE` command issues an attach request for file `DATA_LIBRARY` in the master catalog.
3. Because you included the `STATUS` parameter, you are not notified of any error in processing the `ATTACH_FILE` command. However, you can test the value of the `NORMAL` field for the status variable `LOCAL_STATUS`. This test yields the following information:
 - If the value is `TRUE`, the command has been properly executed and the `IF` statement is not executed.
 - If the value is `FALSE`, an error has occurred in the attempt to process the `ATTACH_FILE` command and the `IF` statement is executed.

Testing for Specific Error Conditions

You can also control error processing by testing for specific error conditions. The `CONDITION` field of the status variable can contain a unique condition code for each diagnostic that NOS/VE issues. In addition, each condition code is associated with a unique condition identifier name.

The `$CONDITION_NAME` function returns a string representation of the name for the condition code identified in the `CONDITION` field. you can compare this value with any specific condition name and control error processing based on the result of the comparison.

For more information about the `$CONDITION_NAME` function, see the NOS/VE Commands and Functions manual. Condition codes and condition names are described in the NOS/VE Diagnostic Messages manual.

Condition Handling

When you specify the `STATUS` parameter on a command, you can alter the command stream based on the completion status of the command. You can also provide condition handlers to alter the command stream in the event of certain system conditions.

The conditions for which you can test are defined by condition categories, and each condition category is identified by a condition name as follows:

Condition Name	Condition Description
<code>PROGRAM_FAULT</code>	The executing program has terminated in error. This includes system-provided programs (for example <code>COBOL</code> or <code>SORT</code>) or your programs.
<code>LIMIT_FAULT</code>	The job resource limit has been reached.
<code>COMMAND_FAULT</code>	The SCL interpreter or a command utility has detected a syntax or semantic error in a command or statement.
<code>ANY_FAULT</code>	Includes any of the fault conditions <code>PROGRAM_FAULT</code> , <code>LIMIT_FAULT</code> , or <code>COMMAND_FAULT</code> .
<code>INTERRUPT</code>	A pause break (terminal interrupt) has been entered.

Condition handling is activated when a condition exists for a command that does not contain a `STATUS` parameter, or is beyond the scope of `STATUS` variable error processing. When condition handling is activated, one of the following events occurs:

- A batch job that has not defined a condition handler for the condition is terminated.
- An interactive job that has not defined a condition handler for the condition terminates the command with a message and prompts you for the next command.
- A condition handler that has been defined for the condition is activated to process the condition.

The severity of a condition determines whether the condition handler is activated. Condition handling takes place as follows:

Severity Level	Condition Handling
INFORMATIVE	Condition handling is not activated; the command that issued the condition completes without aborting, although SCL writes the message to the \$RESPONSE file.
WARNING	Same as for INFORMATIVE.
NONSTANDARD	Same as for INFORMATIVE.
DEPENDENT	Same as for INFORMATIVE.
ERROR	Condition handling is activated.
FATAL	Same as for ERROR.
CATASTROPHIC	Same as for ERROR.

For more information about severity levels, see the NOS/VE Diagnostic Messages manual.

Establishing a Condition Handler (WHEN/WHENEND)

To provide condition handling, use the WHEN/WHENEND statement.

The format of the WHEN/WHENEND statement is as follows:

```
WHEN condition name(s) DO  
    statement list  
WHENEND
```

Condition name(s) can be any of the following keyword values:

- PROGRAM_FAULT
- LIMIT_FAULT
- INTERRUPT
- COMMAND_FAULT
- ANY_FAULT

The WHEN statement is evaluated continually in the block in which it resides. Consequently, if the conditions specified in the WHEN statement occur, the subsequent statement list is executed. If more than one condition handler is established, the most recent condition handler in the current block takes precedence. When a block ceases to exist, any WHEN statements in that block become undefined.

When one of the conditions occurs, two variables are initialized: OSV\$STATUS and OSV\$COMMAND_NAME.

Variable	Description
OSV\$STATUS	Contains a copy of the status record set by the program that determined the condition.
OSV\$COMMAND_NAME	Contains a string whose value is the name of the command that was being processed when the condition occurred.

The WHEN statement is activated, and the statement list can then interrogate these variables in order to determine the error condition and initiate appropriate action.

The following considerations apply to the WHEN/WHENEND block:

- If any of the conditions specified in the WHEN statement occur while the WHEN statement list is being processed, the block that defined the WHEN statement is terminated with an abnormal status.
- Variables created within a WHEN block are local to that block and are released when control is returned from the WHEN statement.
- Exiting from a WHEN block takes place via the CONTINUE statement. If no CONTINUE statement is encountered, the exit occurs after the last statement in the WHEN block is processed.

What happens when you exit from a WHEN block depends on the following factors:

- Condition that caused the WHEN statement to be activated.
- Whether the RETRY option was specified on the CONTINUE statement.

For more information, read *Exiting from a WHEN Block* in the next section.

The following example illustrates how to establish a condition handler. In this case, each time a LIMIT_FAULT condition is encountered, the statements following the WHEN clause are executed. The CHANGE_JOB_LIMIT command increments the current time limit by 60 CP seconds.

```

WHEN limit_fault DO
  put_line ' Incrementing time limit by 60 CP seconds.'
  change_job_limit name=cp_time ..
      resource_limit=($job_limit(cp_time, accumulator)+60)
CONTINUE RETRY
WHENEND

```

Exiting from a WHEN Block (CONTINUE)

The CONTINUE statement transfers control from the WHEN block in which it appears.

The format of the CONTINUE statement is as follows:

```
CONTINUE
  RETRY
  WHEN boolean expression
```

If the boolean expression in the WHEN clause is TRUE, the WHEN block is exited. If the expression is FALSE, the WHEN block is not exited. If the WHEN clause is omitted, the block is exited.

What happens upon exit depends on the following factors:

- The condition that caused the WHEN statement to be activated.
- Whether RETRY was specified on the CONTINUE statement.

The following descriptions illustrate the process of exiting when RETRY is specified on the CONTINUE statement; they also apply to the action taken in the absence of a CONTINUE statement:

COMMAND_FAULT or PROGRAM_FAULT, without RETRY
Processing continues at the statement following the one that caused the WHEN statement to be activated.

COMMAND_FAULT or PROGRAM_FAULT, with RETRY
The statement that caused the WHEN statement to be activated is reprocessed.

INTERRUPT or LIMIT_FAULT, without RETRY
Processing continues at the point of interruption.

INTERRUPT or LIMIT_FAULT, with RETRY
The results are undefined.

The following example establishes a condition handler for a command fault. The user is prompted for several options whenever an `COMMAND_FAULT` condition occurs.

```

WHEN command_fault DO
  create_variable name=response kind=string

  LOOP
    accept_line variable=response ..
    prompt='Enter RETRY, CONTINUE, or command: ' ..
    input=input
    IF $translate(1tu,response) = 'RETRY' THEN
      CONTINUE RETRY
    ELSEIF $translate(1tu,response) = 'CONTINUE' THEN
      CONTINUE
    ELSE
      include_line
    IFEND
  LOOPEND
WHENEND

```

Canceling a Condition (CANCEL)

To cancel one or more condition selections made with a `WHEN` statement, use the `CANCEL` statement. The most recently made `WHEN` statement with the specified condition selection is canceled first.

The format of the `CANCEL` statement is as follows:

```
CANCEL condition(s)
```

The following example cancels the `LIMIT_FAULT` condition established in the previously shown `WHEN` statement:

```
/cancel limit_fault
```

Several conditions can be canceled at the same time, as shown in the next example:

```
/cancel command_fault program_fault
```

Suspending Command Processing (WAIT)

To suspend the processing of commands within your job until a specified time period has elapsed, or until one or more specified events have taken place, use the WAIT command.

The parameters of the WAIT command are as follows:

time, t	: integer 0..4294967295 = \$optional
task_names, task_name, tn	: list of name = \$optional
queue_names, queue_name, qn	: list of name = \$optional
until, u	: key all, any = all
status	: var of status = \$optional

Length of time is measured in milliseconds. For example, to cause a job to wait for 20 seconds (20,000 milliseconds), enter the following form of the command:

```
/wait 20000
```

Alternatively, you can achieve the same results by entering the following form of the WAIT command:

```
/wait 20*1000
```

Changing System Environments

SCL contains statements that enable you to temporarily change system environments within procedures. These are the PUSH and POP statements. You change the system environment by changing one or more environment objects. The following is a list of keyword names for the environment objects you can change with the PUSH and POP statements:

```
COMMAND_LIST  
FILE_CONNECTIONS  
INTERACTION_STYLE  
MESSAGE_LEVEL  
NATURAL_LANGUAGE  
PROGRAM_ATTRIBUTES  
WORKING_CATALOG
```

Establishing a New Environment (PUSH)

To establish a new version of a system environment, use the PUSH statement. The newly established environment remains in effect until you either remove it with a POP statement or exit from the block in which the PUSH statement was executed.

The format of the PUSH statement is as follows:

PUSH keyword(s)

When an environment object is affected by a PUSH statement, its value is copied from the previous version. You can revise or interrogate it with SCL commands, statements, and/or functions specific to that environment object. Only the most recently pushed version of the system environment can be referenced or changed.

The following example illustrates a procedure that temporarily changes the command list environment object. This procedure uses the command library `.AJL.COMMAND_LIBRARY` only during its execution. After exiting from the procedure, you no longer have the command library in your command list.

```
PROC change_environment
  PUSH command_list
  create_command_list_entry e=.ajl.command_library
  :
  "Execute commands found in .ajl.command_library."
  :
PROCEND
```

If you try to PUSH a particular object again within the same block before restoring the original environment with a POP statement, an error results.

Restoring the Previous Environment (POP)

To delete a current version of a system environment object and to restore the changed environment object to its former value, use the POP statement.

The format of the POP statement is as follows:

POP keyword(s)

This deletion operation is performed automatically when your program exits from a block in which the corresponding PUSH statement was executed (see the example in the preceding section).

Writing SCL Procedures and Command Utilities

9

Commands and Functions Used in Procedure Writing	9-3
Reading Lines from a File and Writing Lines to a File	9-10
Reading Lines from a File (ACCEPT_LINE)	9-10
Writing Lines to a File (PUT_LINE, COLLECT_TEXT)	9-12
Inserting Files or Strings into the Command Stream	9-14
Inserting Files (INCLUDE_FILE)	9-14
Using \$COMMAND and \$COMMAND_OF_CALLER	9-15
Interactive Procedures and Utilities	9-15
Inserting Lines (INCLUDE_LINE)	9-16
Inserting Commands (INCLUDE_COMMAND)	9-17
Creating Procedures	9-18
Procedure Calling Environments	9-20
Procedure Format	9-21
Procedure Header Format	9-22
Statement List Format	9-23
Procedure End Format	9-23
Defining Procedure Parameters	9-24
Parameter Name	9-26
Value Specification	9-26
Parameter Data Type	9-27
Formats for the Parameter Data Type	9-27
Data Types	9-28
Using Data Type NAME	9-29
Using Data Type STRING	9-30
Using Data Type INTEGER	9-31
Using Data Type KEY or a KEY Clause	9-31
Using VAR OF and ARRAY OF Clauses	9-32
Value List Type	9-34
Default Specification	9-40
Specifying \$REQUIRED	9-40
Specifying \$OPTIONAL	9-40
Specifying a Value List	9-41
Sample Procedure	9-42
Creating SCL Command Utilities	9-45

- Writing Command Utilities 9-46
 - Overview 9-47
 - Example: Defining a Utility 9-49
 - Example: Subcommand Processors 9-54
 - Processor for CHANGE_FORMAT 9-55
 - Processor for DISPLAY_DATE 9-56
 - Processor for DISPLAY_TIME 9-57
 - Processor for QUIT 9-57
 - Organizing Subcommands and Subcommand Processors 9-58
 - Subcommands 9-58
 - Subcommand Processors 9-59

- Permitting Others to Use Your Utilities 9-60

- Using Your Utility 9-62
 - Executing the Utility 9-62
 - Parameter Prompting 9-64
 - Displays of Subcommand Information 9-65
 - Changing Utility Attributes 9-66

- Formatting SCL Procedures 9-67
 - Formatting Example 9-67
 - Input to the Procedure Formatter 9-69
 - Controlling the Formatting Process 9-70
 - Utility Definition File 9-72
 - Initial List of Utilities and Terminators 9-72
 - Adding Utilities and Terminators to the List 9-75
 - How the Formatter Formats the Input File 9-76
 - How the Formatter Handles Errors 9-80

An SCL procedure consists of a list of statements that are processed when you execute (*call*) the procedure by the name you define for it.

An end user of a procedure sees no difference between calling a procedure and calling any other NOS/VE command. This capability makes it easy to create a new command or to redefine any existing command. Procedure users need not know whether the command is implemented as an SCL procedure or as an executable program.

You can define procedures to accept parameters. Either these parameters can be tested to execute certain commands or statements, or they can be substituted in commands or statements that are executed within the procedure.

A command utility is a command that gives you access to an additional set of commands. NOS/VE not only provides a number of utilities for your use, but also allows you to create your own utilities.¹

1. For a complete list of system-defined utilities, see the discussion of utility definition files later in this chapter.

This chapter discusses the following topics:

- Commands and functions used in procedure and utility writing
- Interactive procedures and utilities
- Creating procedures
- Procedure structure
- Defining procedure parameters
- Creating utilities
- Utility structure
- Defining utility subcommands
- Executing utilities
- Formatting procedures and utilities.

Commands and Functions Used in Procedure Writing

The following tables provide a list of the commands and functions commonly used when writing procedures and a list of all the NOS/VE functions available for your use.

The following commands are discussed in greater detail in the next sections:

INCLUDE_FILE	ACCEPT_LINE
INCLUDE_LINE	PUT_LINE
INCLUDE_COMMAND	COLLECT_TEXT

For more information about the commands and functions listed below, see the NOS/VE Commands and Functions manual.

Table 9-1. Commands Used in Procedure Writing

Command	Description
ACCEPT_LINE	Reads lines from a file into a string variable.
COLLECT_TEXT	Reads lines of text from the command stream and writes them to a specified file.
CREATE_VARIABLE	Creates an SCL variable.
DELETE_VARIABLE	Deletes variable declarations from the current block.
DISPLAY_VALUE	Displays the value of an expression.
DISPLAY_VARIABLE_LIST	Displays the variables available to your current job.
FORMAT_SCL_PROC	Formats a procedure and detects errors in the input file.

(Continued)

Table 9-1. Commands Used in Procedure Writing (Continued)

Command	Description
INCLUDE_COMMAND	Causes the text of a specified string to logically replace the occurrence of the INCLUDE_COMMAND command. This enables you to construct a command through string manipulation and then have the command processed.
INCLUDE_FILE	Inserts into the command stream a text file containing SCL statements.
INCLUDE_LINE	Inserts into the command stream a string containing SCL statements.
PUT_LINE	Writes lines from a string variable to a file.
WAIT	Suspends command processing until a specified number of milliseconds have elapsed, or until another specified event or set of events has taken place.

Table 9-2. Topical List of NOS/VE Functions

Topic	Function Name
Accessing Date and Time:	
Accessing the current date	\$DATE
Accessing the current time	\$TIME
Converting Data:	
Converting a string to a file name	\$FNAME
Converting a string or a boolean to an integer	\$INTEGER
Converting a string to a name	\$NAME
Converting a name, file, or variable to a string	\$STRING
Converting any value to a string	\$STRREP
Converting a string to a variable name	\$VNAME
Returning Constant Values:	
Returning the maximum integer value	\$MAX_INTEGER
Returning the minimum integer value	\$MIN_INTEGER
Returning the maximum name length	\$MAX_NAME
Returning the maximum string length	\$MAX_STRING
Returning the maximum number of value sets	\$MAX_VALUE_SETS
Returning the maximum number of values	\$MAX_VALUES

(Continued)

Table 9-2. Topical List of NOS/VE Functions *(Continued)*

Topic	Function Name
Manipulating Strings:	
Finding the character given the ordinal	\$CHAR
Finding the ordinal given the character	\$ORD
Determining string length	\$STRLEN
Changing characters in a string	\$TRANSLATE
Copying a string and adding apostrophes	\$QUOTE
Deleting trailing spaces from a string	\$TRIM
Searching for a character in a set of characters	\$SCAN_ANY
Searching for a character not in a set of characters	\$SCAN_NOT_ANY
Searching one string for another string	\$SCAN_STRING
Processing substrings	\$SUBSTR
Obtaining File and Catalog Information:	
Determining the access mode of a file	\$ACCESS_MODE
Interrogating file attributes	\$FILE
Accessing the working catalog name	\$CATALOG
Interrogating file paths	\$PATH

(Continued)

Table 9-2. Topical List of NOS/VE Functions (Continued)

Topic	Function Name
Obtaining Hardware Information:	
Returning attributes of the mainframe	\$MAINFRAME
Returning attributes of the processor	\$PROCESSOR
Obtaining Information about Jobs:	
Interrogating job resource limits	\$JOB_LIMIT
Interrogating job attributes	\$JOB
Determining the status of a job	\$JOB_STATUS
Determining the system default values for a job's attributes	\$JOB_DEFAULTS
Determining the terminal model	\$TERMINAL_MODEL
Interrogating remote validation	\$REMOTE_VALIDATION
Determining your interaction style	\$INTERACTION_STYLE
Determining the name of a task	\$TASK_NAME
Determining the status of a task	\$TASK_STATUS
Determining the message level	\$MESSAGE_LEVEL
Determining the language of messages	\$NATURAL_LANGUAGE
Determining the execution ring	\$RING
Interrogating variable attributes	\$VARIABLE
Testing job default program attributes	\$PROGRAM

(Continued)

Table 9-2. Topical List of NOS/VE Functions *(Continued)*

Topic	Function Name
Obtaining Information about Output Files:	
Interrogating output file attributes	\$JOB_OUTPUT
Determining the status of an output file	\$OUTPUT_STATUS
Functions Dealing with Status Values:	
Determining the condition code	\$CONDITION_CODE
Determining the condition name	\$CONDITION_NAME
Determining the severity of a condition	\$SEVERITY
Constructing a status value	\$STATUS
Obtaining the completion status of a command	\$PREVIOUS_STATUS
Parameter Reference Functions:	
Determining whether a parameter is passed	\$SPECIFIED
Determining the number of value sets	\$SET_COUNT
Determining the number of value elements in a value set	\$VALUE_COUNT
Determining whether parameters are specified as ranges	\$RANGE
Determining the kind of value of a parameter	\$VALUE_KIND
Determining or setting a parameter value	\$VALUE
Accessing the entire value list	\$PARAMETER
Accessing the entire parameter list	\$PARAMETER_LIST

(Continued)

Table 9-2. Topical List of NOS/VE Functions *(Continued)*

Topic	Function Name
Miscellaneous Functions:	
Interrogating utility attributes	\$UTILITY
Calculating the modulus of two integers	\$MOD
Generating unique names	\$UNIQUE
Determining the source of a command	\$COMMAND_ SOURCE
Reserved for future use	\$QUEUE

Reading Lines from a File and Writing Lines to a File

Commands are provided to read a line from a file into a string variable and to write a string variable to a file.

Reading Lines from a File (ACCEPT_LINE)

To read lines from a file into a string variable, use the ACCEPT_LINE command. You must create the string variable before entering the ACCEPT_LINE command.

To guarantee that the ACCEPT_LINE command reads from a terminal, do not use a standard file (such as \$INPUT) for the INPUT parameter. Instead, use a file such as the job file INPUT. This guarantees that reading occurs from the terminal rather than from another target file connected to the \$INPUT file. It also ensures that the prompt is written to the terminal.

In the example on the following page, the WRITE_FILE_TO_OUTPUT procedure reads each line from file INFILE into a string variable named INPUT_STRING until the end-of-information is reached.

By default, NOS/VE commands which access a file perform an implicit attach/detach of the file. For example, the ACCEPT_LINE command performs an automatic attach before reading a line and an automatic detach after reading a line from a file. In the example on the next page, however, the WRITE_FILE_TO_OUTPUT procedure eliminates the need for this by using ATTACH_FILE to explicitly attach the file. The \$ASIS file position maintains the correct file position for each execution of the ACCEPT_LINE command. After the file is read, it is explicitly detached with the DETACH_FILE command.

File INFILE is assumed to have the following contents:

```
123
 456
 789
```

File `WRITE_FILE_TO_OUTPUT` has the following contents:

```
PROC write_file_to_output(
  input,i : file = $required
  output,o : file = output
  status   : var of status = $optional
)

  create_variable input_string kind=string
  create_variable input_count kind=integer
  attach_file f=$value(input)
  LOOP
    accept_line variable = input_string ..
      input = $fname($string($value(input))//'.$asis') ..
      prompt = ' >>>?' line_count=input_count.
    EXIT WHEN input_count=0
    display_value input_string
  LOOPEND
  detach_file f=$value(input)

PROCEND write_file_to_output
```

When the procedure is executed, the following output is produced:

```
/write_file_to_output input=infile
123
  456
    789
```

Writing Lines to a File (PUT_LINE, COLLECT_TEXT)

To write lines from a string to a file, use the following commands. A list of each command's parameters is included.

- PUT_LINE

lines, line, l	: list of string = \$required
output, o	: file = \$output
status	: var of status = \$optional

- COLLECT_TEXT

output, o	: file = \$required
until, u	: string = '**'
prompt, p	: string 0..30 = 'ct? '
substitution_mark, sm	: string 1 or key none = none
status	: var of status = \$optional

If you are writing more than one line in succession to a file, use the COLLECT_TEXT command for faster response. This command opens the output file only once for multiple lines, whereas the PUT_LINE command opens the file for each line. In addition, the COLLECT_TEXT command allows you to substitute the values of variables and string expressions in the output file.

The following example writes a three-line message to file OUTPUT. The leading space in each of the three strings is a format effector (a space character) that causes each string to print on a separate line.

```
/put_lines lines=( ..  
../' Today's date:  '$date(month) ..  
../' The current time:  '$time(ampm) ..  
../' Welcome to NOS/VE.' )  
Today's date:  March 28, 1988  
The current time:  2:45 PM  
Welcome to NOS/VE.
```

The following command sequence creates a file named DATE into which an SCL procedure is entered and then called.

```

/collect_text output=date until='end'
ct? proc date ( )
ct? display_value $date(month)
ct? procend
ct? end
/date      "Execute the procedure here."
March 28, 1988

```

NOTE

Although you can use both the PUT_LINE and DISPLAY_VALUE commands to display the value of one or more strings, observe the following differences:

- The PUT_LINE command never adds a page title or format effectors (the string is assumed to contain any necessary format effectors).
 - Depending on the attributes of the output file, the DISPLAY_VALUE command may add a page title or format effectors. Format effectors are discussed in chapter 6, Job Management.
-

Inserting Files or Strings into the Command Stream

NOS/VE provides commands that insert (include) a file or string into the command stream. The following commands are described in this section:

```
INCLUDE_FILE  
INCLUDE_LINE  
INCLUDE_COMMAND
```

Inserting Files (INCLUDE_FILE)

To cause a text file containing SCL statements to logically replace the occurrence of the command itself, use the INCLUDE_FILE command. The commands in the specified file are processed in the same context as the INCLUDE_FILE command. That is, the command has access to the same variables and conditions, as well as to parameter substitution.

To understand the example that follows, assume that file PROLOG contains the following statement list:

```
display_value 'Login at '//$time(ampm)//' on '//..  
$date(month)//'.'
```

Entering the following INCLUDE_FILE command produces the indicated results:

```
/include_file file=prolog  
Login at 3:21 PM on March 28, 1987.
```

The inserted statement list is treated the same as the statement list of an unlabeled BLOCK statement.

Using \$COMMAND and \$COMMAND_OF_CALLER

NOS/VE uses \$COMMAND and \$COMMAND_OF_CALLER to specify an active command stream and the current position within it.

- \$COMMAND specifies the position in the command stream from which the SCL interpreter is currently reading input. When you call a procedure, the interpreter begins reading from the procedure file. \$COMMAND in this case specifies the position between the PROC and PROCEND statements from which input is currently being read. If you insert \$COMMAND into the procedure's statement list using the INCLUDE_FILE command, the interpreter considers the current input file to be the statement list following the INCLUDE_FILE command up to the PROCEND statement.
- \$COMMAND_OF_CALLER specifies the \$COMMAND file from which the requesting procedure was called. That is, while a procedure is executing, \$COMMAND_OF_CALLER specifies the current position in the command stream which called the procedure.

Interactive Procedures and Utilities

You can create procedures and utilities which allow continuous interactive input in two ways:

- By inserting \$COMMAND_OF_CALLER into the statement list of a procedure or utility using the INCLUDE_FILE command.
- By passing \$COMMAND to a procedure or utility as a parameter value and then inserting the parameter value into the statement list of the procedure or utility using the INCLUDE_FILE command.

Either instance allows continuous interactive input to the procedure or utility provided the call to the procedure or utility is issued from an interactive terminal.

The interpreter begins reading from the \$COMMAND file which called the procedure or utility at the point following the call. The remaining statements in the procedure or utility block cannot be executed until you exit the block created by the INCLUDE_FILE command, for example via an EXIT statement.

Inserting Lines (INCLUDE_LINE)

To cause the text of a string to logically replace the occurrence of the command itself, use the INCLUDE_LINE command. The commands in the specified string are processed in the same context as the INCLUDE_LINE command. That is, the string has access to the same variables, conditions, and parameter substitution as the INCLUDE_LINE command.

This command lets you construct a line of one or more statements via string manipulation and then process it.

The following example uses string concatenation to create a line and then processes the line with the INCLUDE_LINE command:

```
/time_call = '$time(ampm)'  
/date_call = '$date(month)'  
/command_list = 'display_value ('//time_call//' '//date_call//)'  
/include_line sl=command_list  
3:26 PM  
March 28, 1987
```

The inserted statement list is treated as the statement list of an unlabeled BLOCK statement.

You can use the INCLUDE_LINE command to control any syntax errors encountered during evaluation of assignment statements. For example, if you enter the following commands, the error shown is displayed:

```
/create_variable n=address k=string  
/address = 1304  
--ERROR-- Wrong kind of value for variable.  
/
```

Assigning an integer value to a string variable is a syntax error. Since assignment statements do not have status parameters, you cannot perform error processing for errors of this kind.

The following example demonstrates the use of the INCLUDE_LINE command to control syntax errors in assignment statements:

```
/create_variable n=incl_status k=status  
/include_line sl='address = 1304' status=incl_status  
/display_value ('This error occurred:' incl_status)  
This error occurred:  
--ERROR-- Wrong kind of value for variable.
```

Inserting Commands (INCLUDE_COMMAND)

To construct a command through string manipulation and then process the command, use the `INCLUDE_COMMAND` command. The command causes the text of the specified string to logically replace the occurrence of the `INCLUDE_COMMAND` command. The command in the specified string is processed in the same context as the `INCLUDE_COMMAND` command. That is, the command in the string has access to the same variables, conditions, and parameter substitutions as the `INCLUDE_COMMAND` command.

Be aware of the following distinctions between the `INCLUDE_LINE` command and the `INCLUDE_COMMAND` command:

- The `INCLUDE_LINE` command permits processing of multiple commands and control statements. However, the `INCLUDE_COMMAND` command accepts only one command.
- The `INCLUDE_LINE` command creates an input control block; that is, it treats the statement list as if it were the statement list of an unlabeled `BLOCK` statement. The `INCLUDE_COMMAND` command does not.

The following example creates a string that is to be interpreted as a command. Next, it processes the command with the `INCLUDE_COMMAND` command.

```
/time_call='$time(ampm)'
/command='display_value ('//time_call//')'
/include_command command=command
11:41 AM
```

Creating Procedures

To create a procedure, complete the following steps (it is assumed that your working catalog is set to \$USER):

1. Use the EDIT_FILE utility to create a file to contain the procedure. For example, the following command creates file \$LOCAL.SAMPLE_PROC to hold a procedure:

```
/edif $local.sample_proc
```

For further information about the EDIT_FILE utility, see the NOS/VE File Editor manual.

2. Create the procedure. Details about procedure format are found later in this chapter. The following is a sample procedure:

```
proc print_report, prir (  
  file, f : file = $required  
  number, n : integer 1..10 = 5  
  banner, b : string = 'MONTHLY STATUS REPORT'  
  status)  
print_file file=$value(file) copies=$value(number) ..  
  routing_banner=$value(banner) ..  
  comment_banner=$strep($value(number))// ' copies printed.'  
procend print_report
```

3. Format the procedure using the SCL formatter. This is an optional step you can perform to make your procedure easier to read. For example, the following command formats the procedure contained in file `$LOCAL.SAMPLE_PROC`:

```
/format_scl_proc input=$local.sample_proc ..
../output=prir
```

Because you are writing it to a permanent file (located in the working catalog `$USER`, which contains only permanent files), the file `$USER.PRIR` containing the procedure is now permanent.

The formatted procedure appears as follows:

```
PROC print_report, prir (
  file, f   : file = $required
  number, n : integer 1..10 = 5
  banner, b : string = 'MONTHLY STATUS REPORT'
  status   : var of status = $optional
)

  print_file file=$value(file) copies=$value(number) ..
  routing_banner=$value(banner) ..
  comment_banner=$strrep($value(number))/' copies printed.'

PROCEND print_report
```

4. Execute the procedure. You can execute procedures in two ways.
 - Enter the name of the file containing the procedure.
 - Enter the name of the procedure which resides on an object library.

For more information, see chapter 5, Command and SCL Procedure Execution.

Procedure Calling Environments

SCL procedures can be called from the following environments:

- From outside a procedure.
- From within another SCL procedure. These are called *nested* calls.
- From within itself. These are called *recursive* calls.

Access to variables in procedures is discussed in chapter 7, SCL Variables, Types, and Expressions.

SCL procedures cannot be defined within another SCL procedure (nested definitions). For example, the following structure is not valid:

```
PROC a
  :
  PROC b
    :
    PROCEND b
  :
PROCEND a
```

Procedure Format

This section describes the format of SCL procedures.

An SCL procedure has the following general format:

```
PROC procedure names (parameter definitions)  
      statement list
```

```
PROCEND procedure name
```

The procedure header is required and always begins with a PROC statement. The procedure end is required and always begins with a PROCEND statement. The statement list is optional.

For example, assume the file PROC_FILE contains the procedure DISPLAY_NUMBER as follows:

```
proc display_number ( )  
  :  
  statement list  
  :  
procend
```

The procedure header, statement list, and procedure end are described in more detail in the following paragraphs.

Procedure Header Format

The procedure header has the following format:

```
PROC procedure name,alias,...,alias
```

The following considerations apply to the procedure header:

- Minimally, it must include a PROC statement with one procedure name. A procedure name is any valid SCL name.
- The PROC statement must be separated from the procedure name by a space.
- The procedure header may also include aliases for the procedure name. Supply the aliases in a list in which each name is separated by a comma or a space.

The following procedure header defines a PROC statement with the procedure name DISPLAY_NUMBER:

```
PROC display_number
```

The DISPLAY_NUMBER procedure name shown in the following example contains the aliases DISPLAY_NUMBERS and DISN:

```
PROC display_number,display_numbers,disn
  :
  statement list
  :
PROCEND display_number
```

With this definition, any of the following calls are valid provided the procedure has been added to the command list:

```
display_number
display_numbers
disn
```

Statement List Format

A statement list constitutes the body of a procedure. The statement list can contain any valid SCL statement, including SCL comments. For example:

```
PROC display_number,display_numbers,disn
    "This procedure displays the number 3."
    display_value 3

PROCEND display_number
```

An SCL procedure constitutes its own block. When the procedure terminates, the block is undefined. Any variables created in the procedure cease to exist, and outstanding calls to utilities or procedures are terminated.

Procedure End Format

Optionally, you can specify the name of the procedure on the procedure end statement PROCEND in the following format:

```
PROCEND procedure name
```

If you specify the procedure name, it must match the first name listed on the procedure header as in the following example:

```
PROC display_number, display_numbers, disn
    :
PROCEND display_number
```

In general, include the procedure name in the PROCEND statement for documentation purposes.

Defining Procedure Parameters

When you define a procedure, you can optionally specify that it is to accept parameters. As with parameters on NOS/VE commands, the parameters you specify allow you to pass data in and out of the procedure. The following is a list of some common uses for parameters:

- Specifying the source of input for the command or procedure.
- Specifying the destination for output from the command or procedure.
- Passing variables in and out of the command or procedure.
- Providing the specific data for the purposes of the command or procedure.
- Providing a record of the command or procedure's completion status.

Recall that the definition of a procedure has the following format:

```
PROC procedure names (parameter definitions)  
      statement list
```

```
PROCEND procedure name
```

The following guidelines apply to this procedure format:

- The opening and closing parentheses are required.
- The opening parenthesis must be on the same line as the procedure names.
- Each parameter definition can be expressed in one of the following ways:
 - On separate lines:

```
parameter definition  
parameter definition
```
 - More than one to a line, separated by semicolons:

```
parameter definition; parameter definition
```

- Split over two or more lines. In this case, an ellipsis (..) must appear at the end of each line to be continued:

```
parameter ..
definition
```

The following is a sample format of a procedure header with parameter definitions included:

```
PROC procedure names (
parameter definition
:
parameter definition)
```

Each parameter definition has the following format:

```
parameter names:value specification=default specification
```

parameter names

Name or names by which you can reference a parameter. For each parameter you define, you must include at least one name.

value specification

Kind of value and whether you can represent it as a list and/or a range. You can define lists further to allow a specific number of value sets. In addition, you can define each value set to contain a specific number of value elements. The value specification is optional.

default specification

Indicates whether a parameter is required or optional. If the parameter is optional, you can also specify its default value. The default specification is optional.

In the procedure definition, the full parameter definition format appears as follows:

```
PROC procedure names (
parameter names:value specification=default specification)
statement list
```

```
PROCEND procedure name
```

Parameter names, the value specification, and the default specification are described in more detail in the following sections.

Parameter Name

A parameter name enables you to reference a particular parameter in the procedure header. For each parameter you define, you must include at least one name. The following example illustrates how you can write the `DISPLAY_NUMBER` procedure to accept a number:

```
PROC display_number,display_numbers,disn (  
    number,numbers,n:integer)
```

The `NUMBER` parameter is defined with two aliases and is given a value specification of integer (value specifications are described in the next section).

Value Specification

The value specification specifies the type of value and whether you can represent it as a list and/or range. The value specification consists of one of the following elements:

Value Specification	Description
data type	Specifies the type of value the parameter can be.
value list type OF data type	Specifies whether the parameter value can be given as a list and/or a range of values.

The value specification is optional. If you do not specify a data type, the system assumes the parameter is a value of type `FILE`. An exception occurs if the parameter name is `STATUS`; in this case, the data type is assumed to be `VAR OF STATUS`.

The following sections describe the two kinds of value specifications in more detail.

Parameter Data Type

The parameter's data type defines:

- The type of value.
- Whether it is a variable or an array.
- Whether it can be represented by one or more keywords.

Formats for the Parameter Data Type

The following are valid formats for the parameter data type:

data type

KEY keyword

data type OR KEY keyword

The following are valid parameter formats when the data type is a valid variable type:

VAR OF data type

ARRAY OF data type

Data Types

The following table lists the data types:

Data Type	Description
FILE	File
NAME	SCL name
STRING	String
INTEGER	Integer
REAL	Real number
BOOLEAN	Boolean value
KEY	Keyword value
STATUS	Status variable
ANY	Entry indicating that any data type can be used. (You can use the \$VALUE_KIND function to evaluate the parameter and return the actual data type.)
application value name	Name of an application value. For further information on application values, see the CYBIL System Interface manual.

The following paragraphs describe how to use the data types.

Using Data Type NAME

When using the data type NAME, you can specify its minimum and maximum character lengths in either of the following formats:

NAME length

NAME minimum length..maximum length

If you omit the length, the system assumes 1..\$MAX_NAME (31 characters)

The following examples create specific name lengths:

name 10	Name consisting of exactly 10 characters.
name 1..9	Name consisting of 1 to 9 characters.
name 2..2	Name consisting of exactly 2 characters.

Using Data Type STRING

When using the data type **STRING**, you can specify its minimum and maximum character lengths in either of the following formats:

STRING length

STRING minimum length..maximum length

If you omit the string length, the system assumes **0..\$MAX_STRING** (256 characters) string length.

The following examples indicate strings of specific lengths:

string 10 String consisting of exactly 10 characters.

string 1..40 String consisting of 1 to 40 characters.

string 1..2 String consisting of 1 or 2 characters.

Using Data Type INTEGER

When using the data type `INTEGER`, you can specify its minimum and maximum integer values as follows:

`INTEGER minimum value..maximum value`

If you omit the minimum and maximum values, the system assumes `$MIN_INTEGER..$MAX_INTEGER`
 (-9,223,372,036,854,7,775,808..+9,223,372,036,854,7,775,807)

The following examples illustrate how to represent integer values:

`integer 1..100` Integer from 1 to 100.

`integer 0..$max_integer` Integer from 0 to `$MAX_INTEGER`.

Accordingly, you can define the `DISPLAY_NUMBER` procedure to accept numbers only in the range -100 to +100:

```
PROC display_number,display_numbers,disn (
  number,numbers,n : integer -100..100 = $required
  status           : var of status = $optional
)
```

Using Data Type KEY or a KEY Clause

When using the data type `KEY` or a `KEY` clause, you can accept one of a list of names you define as a parameter value. For example, assume that a parameter named `COUNT` specifies the number of lines of a file to display. You can define the `COUNT` parameter as accepting the keywords `ALL` or `NONE`. Subsequently, specifying `ALL` causes the system to list all lines of the file in question. You can define such a parameter as follows:

`count,c : key all none`

If you want to permit an integer value and a keyword, define the `COUNT` parameter as follows:

`count,c : integer or key all none`

Using VAR OF and ARRAY OF Clauses

If you want the value of the parameter passed to be a variable, or if you want the procedure to return a value, use the VAR OF clause. If you want a variable array with a dimension greater than 1 to be passed to the procedure, use the ARRAY OF clause.

When using a VAR OF or ARRAY OF clause, you must also specify one of the following data types for the variable:

STRING
INTEGER
REAL
BOOLEAN
STATUS
ANY

Specifying ANY for the value kind allows a variable of value kind STRING, INTEGER, REAL, BOOLEAN, or STATUS. You can use the \$VALUE_KIND function to evaluate the variable and return the variable's kind.

In general, the following considerations apply to accessing parameter values from within a procedure:

- Values supplied as parameters cannot be directly accessed within the procedure. The \$VALUE function must be used to access the procedure's parameter values.
- Parameter values defined by the ARRAY OF clauses cannot be directly manipulated within the procedure. A copy of the variable or array must be implicitly created and its value reassigned to the parameter using the \$VALUE function.
- You may not use a KEY clause in conjunction with a VAR OF or ARRAY OF clause.
- You may not use a value list type, described in the next section, in conjunction with a VAR OF or ARRAY OF clause.

The following example illustrates one way that values can be assigned to parameter values which are variables.

```
PROC ask (
  answer, a : var of string = $required
  status    : var of status = $optional
)

  input_answer = ''
  REPEAT
    accept_line v=input_answer i=input ..
      p='Enter Yes or No - '
      input_answer = $substr($translate..
        (1tu input_answer) 1 1)
  UNTIL (input_answer = 'Y' OR ..
    input_answer = 'N')
  $value(answer) = input_answer

PROCEND ask
```

The procedure reflects the following characteristics:

- The single parameter, ANSWER, is a string variable.
- The procedure issues a prompt until you enter a value starting with Y, y, N, or n.
- The procedure then sets the value of the ANSWER parameter to Y or N.

The following example uses the procedure:

```
/response=''
/ask response
Enter Yes or No - you bet
/display_value response
Y
```

For an example of a procedure that returns values as variables and in an array variable, see the online Examples manual (the name of the example is SET_PARAMETERS). For more information about the \$VALUE and \$VALUE_KIND functions, see the NOS/VE Commands and Functions manual.

Value List Type

If you want a parameter to accept a list of value sets or a range, you need to include the value list type in the parameter definition. The value list type defines the parameter as a list of value sets.

The following are valid formats for the value list type:

LIST

LIST value set count

LIST value set count, value count

LIST value set count, RANGE

LIST value set count, value count, RANGE

LIST RANGE

RANGE

Element	Description
value set count	<p>Entry that identifies the number of value sets allowed. The value set count consists of one of the following:</p> <ul style="list-style-type: none"> • Integer expression (indicating both the minimum and maximum number of value sets). • Range of integer expressions (indicating the minimum to maximum number of value sets). <p>When you specify <i>LIST</i>, the default value is the value of <code>\$MAX_VALUE_SETS</code>. When you do not specify <i>LIST</i>, the default value is 1.</p>
value count	Entry that specifies the number of elements in a value set.
RANGE	Entry indicating that any value element can be given a range of values.

The examples on the following pages illustrate how value specifications function within a procedure.

Example 1: Using a List of Value Sets.

Assume that you create the following procedure:

```
PROC display_number,display_numbers,disn (
  number,numbers,n : list 1..10 of integer = $required
  status           : var of status = $optional
)

  FOR i = 1 TO $set_count(number) DO
    display_value $strrep($value(number,i),10)
  FOREND

PROCEND display_number
```

The procedure reflects the following characteristics:

- It displays the decimal value of one or more integer expressions.
- The value list type specifies that the NUMBER parameter can be represented by 1 to 10 value sets of type integer.
- The NUMBER parameter is required and has the aliases NUMBERS and N.
- The \$SET_COUNT function returns a count of the number of value elements for the NUMBER parameter.
- The \$VALUE function returns the value of the NUMBER parameter.

After you add the procedure to the command list so that it can be referenced by any of its procedure names, the following calls are valid:

```
/display_number 2+3           One value set.
5
/disn (0a(16),100(8)+100(2))  Two value sets.
10
68
/display_numbers n=(1,2,3,4)   Four value sets.
1
2
3
4
```

Example 2: Including a Range in a List of Value Sets

Assume that you change the procedure in example 1 to include an IF statement:

```

PROC display_number,display_numbers,disn (
  number,numbers,n : list 1..10, range of integer = $required
  status           : var of status = $optional
)

FOR i = 1 TO $set_count(number) DO
  IF $range(number,i) THEN
    FOR j = $value(number,i,1,low) TO ..
      $value(number,i,1,high) DO
        display_value $strrep(j,10)
      FOREND
    ELSE
      display_value $strrep($value(number,i),10)
    IFEND
  FOREND

PROCEND display_number

```

This DISPLAY_NUMBER procedure tests whether the NUMBER parameter was specified as a range, and then determines the lower and upper limits of the range. The following activity takes place:

1. The \$RANGE function tests a value element for the NUMBER parameter. If the value element is specified as a range, \$RANGE returns a TRUE value.
2. The \$SET_COUNT function returns a count of the number of value elements for the NUMBER parameter.

3. The \$VALUE function returns the value of the parameter.

The \$RANGE, \$SET_COUNT, and \$VALUE functions are described in the NOS/VE Commands and Functions manual.

The following call to procedure DISPLAY_NUMBER is now valid:

```
/display_number (1..3,25(16),10..11,-10,-1..1)
1
2
3
37
10
11
-10
-1
0
1
```

In the example, the following five value sets are passed to the procedure:

Value Set	Value Elements
1..3	The range of numbers 1 through 3.
25(16)	The number 25 (hexadecimal).
10..11	The range of numbers 10 through 11.
-10	The number -10.
-1..1	The range of numbers -1 through +1.

Example 3: Including a Value Count in a List of Value Sets

By including a value count, you indicate that each value set can contain one or more value elements. Specify the value count either as an integer expression (indicating both the minimum and maximum number of value elements) or as a range of integer expressions (indicating the minimum to maximum number of value elements). The default value count is 1.

Assume that you change the procedure in example 2 as follows:

```

PROC display_number,display_numbers,disn (
  number,numbers,n : list 1..10, 1..2, ..
    range of integer = $required
  status           : var of status = $optional
)

FOR i = 1 TO $set_count(number) DO
  IF $value_count(number,i) = 1 THEN
    base=10
  ELSE
    base=$value(number,i,2)
  IFEND
  IF $range(number,i) THEN
    FOR j = $value(number,i,1,low) TO ..
      $value(number,i,1,high) DO
        display_value $strrep(j,base)
      FOREND
    ELSE
      display_value $strrep($value(number,i),base)
    IFEND
  FOREND
PROCEND display_number

```

In this procedure, the following activity takes place:

- The NUMBER parameter is defined as a list of 1 to 10 value sets, each of which can contain 1 or 2 value elements.
- The \$VALUE_COUNT function returns a count of the number of value elements in the NUMBER parameter.
- In this instance, the first value element specifies the integer expression to be displayed; the second value element specifies the base in which the number is to be displayed. If you omit the base, the system assumes decimal.

The \$VALUE_COUNT, \$SET_COUNT, \$RANGE, and \$VALUE functions are described in the NOS/VE Commands and Functions manual.

The following is a valid call to procedure DISPLAY_NUMBER:

```
/disn n=( (100(8),16) 2**4 (4096,8) )
40
16
10000
```

In the preceding call, the NUMBER parameter is represented by the following three value sets:

Value Set	Value Elements
-----------	----------------

(100(8),16)	The number 100 octal in base 16.
-------------	----------------------------------

2**4	The number 2**4 in base 10.
------	-----------------------------

(4096,8)	The number 4096 in base 8.
----------	----------------------------

Default Specification

The default specification indicates whether you want a parameter to be required or optional. Include the default specification in the procedure in the following format:

```
parameter names : value specification = default specification
```

You can define the default specification with one of the following entries:

```
$REQUIRED  
$OPTIONAL  
value list
```

The default specification is optional. Omitting the default specification is equivalent to specifying `$OPTIONAL`.

The three entries are described in the following sections.

Specifying `$REQUIRED`

If you specify `$REQUIRED` as the default specification, you must enter the parameter when calling the procedure. If interactive prompting is in effect, you are prompted for the parameter; otherwise SCL terminates the call with a diagnostic message.

The following parameter definition specifies a required parameter:

```
output,o: file = $required
```

Specifying `$OPTIONAL`

If you specify `$OPTIONAL` as a default specification, you can omit the parameter from the procedure call. The system assumes no default value for the parameter. The following parameter definition specifies an optional parameter:

```
output,o: file = $optional
```

Specifying a Value List

If you specify a value list as the default specification, the parameter is optional. If you omit the parameter from the procedure call, the system assumes the default value specified by the value list. The default value is treated exactly as if it had been supplied in the parameter list, with one exception: using the \$SPECIFIED function in the body of the procedure produces a FALSE result. For more information about the \$SPECIFIED function, see the NOS/VE Commands and Functions manual.

In the following example, the DISPLAY_NUMBER procedure is defined to accept an OUTPUT parameter with a default value of \$OUTPUT:

```
PROC display_number,display_numbers,disn (
  number,numbers,n : list 1..10, 1..2, ..
  range of integer -100..100 = $required
  output,o          : file = $OUTPUT
  status           : var of status = $optional
)
```

If you omit the OUTPUT parameter on the call to the DISPLAY_NUMBER procedure, the output from the procedure is written to file \$OUTPUT.

Sample Procedure

Figure 9-1 contains a procedure that accepts parameters. These parameters:

- Identify the printed file listing using the `ROUTING_BANNER` parameter.
- Specify the number of copies to be printed.

The procedure then prints a comment on the banner page indicating how many copies were requested.

To call the procedure, enter the name of the procedure (`PRINT_REPORT`), followed by the parameter entries defined in the procedure itself. This step assumes that you have already placed the procedure in an object library and added the library to the command list. For more information about executing procedures, see chapter 5, *Command and SCL Procedure Execution*.

The following call instructs `NOS/VE` to print four copies of file `ACCOUNTING_STATS` with the header `WEEKLY STATUS REPORT` on the first page of each listing:

```
/print_report file=accounting_stats number=4 ..  
../banner='WEEKLY STATUS REPORT'
```

```

① PROC print_report, prir (
  ② file, f   : file = $required
    number, n : integer 1..10 = 5
    banner, b : string = 'MONTHLY STATUS REPORT'
    status    : var of status = $optional
  ③ )

  ④ print_file file=$value(file) copies=$value(number) ..
    routing_banner=$value(banner) ..
    comment_banner=$strrep($value(number))// ' copies printed.'

  ⑤ PROCEND print_report

```

Figure 9-1. Sample SCL Procedure

The following sequence describes what occurs in the sample procedure:

- ① Procedure header (PROC statement) identifies the procedure name PRINT_REPORT and its alias PRIR. The opening parenthesis (which must be on the same line as the procedure name) introduces the parameter list.
- ② Parameters are defined on separate lines.
 - The FILE parameter specifies the file to be printed. When entered, this parameter can be abbreviated to F. This is a required parameter.
 - The NUMBER parameter specifies the number of copies to be printed. SCL accepts only an integer value from 1 through 10 for this parameter. This parameter is optional. If you do not specify this parameter when calling the procedure, SCL assumes you want five copies printed.
 - The BANNER parameter specifies the text of the comment you want printed on the first page of each printout. SCL accepts only a string value for this parameter. This parameter is optional. If you do not specify this parameter when calling the procedure, the value MONTHLY STATUS REPORT is used.

- ③ A closing parenthesis terminates the parameter definitions.
- ④ The PRINT_FILE command is configured so that it can accept the parameters specified in the procedure call. When the procedure is called, this command is executed with the designated parameters in effect.
 - The FILE parameter accepts the value contained in the FILE parameter that must accompany the procedure call. The \$VALUE function performs the necessary reference.
 - The COPIES parameter accepts the value contained in the optional NUMBER parameter that accompanies the procedure call. The \$VALUE function performs the necessary reference.
 - The ROUTING_BANNER parameter accepts the value contained in the optional BANNER parameter that accompanies the procedure call. The \$VALUE function performs the necessary reference.
 - The expression for the COMMENT_BANNER parameter does three things:
 - Accepts the value contained in the optional NUMBER parameter that accompanies the procedure call.
 - Converts that value to a string representation.
 - Concatenates that string representation with the string ' copies printed.'

Values supplied as parameters cannot be directly accessed within the procedure. The \$VALUE function must be used to access the procedure's parameter values. The \$STRREP function performs the necessary value conversions.

The \$VALUE and \$STRREP functions are described in the NOS/VE Command and Function manual.

- ⑤ The PROCEND statement terminates the procedure.

Creating SCL Command Utilities

A *command utility* is a command that gives you access to an additional set of commands. For example, NOS/VE provides a command utility named `CREATE_OBJECT_LIBRARY`. When you enter the command `CREATE_OBJECT_LIBRARY`, a set of commands for editing files is added to your *command list*.²

Because these commands are available only while the utility is active, they are called *subcommands* of the utility. These commands perform the actual operations of the utility. Once you exit the utility, the subcommands are removed from your command list and are no longer available.

While the command utility is in effect, you can continue to enter all the SCL commands, functions, and control statements in your command list as well as the utility's subcommands.

In addition to providing system-defined utilities such as the `CREATE_OBJECT_LIBRARY` utility, SCL allows you to define your own utilities.³ The next sections describe the following procedures:

- How to write a command utility.
- How to permit others to use your utility.
- How to use your utility.

2. For a more complete discussion of command lists, see chapter 5, Command and SCL Procedure Execution

3. For a complete list of system-defined utilities, see the discussion of utility definition files later in this chapter.

Writing Command Utilities

The next sections describe how to write a utility:

- The Overview section describes the overall process involved in writing a utility. The following topics are addressed:
 - Creating a procedure file to contain the utility.
 - Formatting the procedure file.
 - Adding the utility to your command list.

It does not describe in detail the actual defining of the utility.

- Example: Defining a Utility presents an example of a utility in detail as one way you can define a utility.
- Organizing Subcommands and Subcommand Processors expands the description of subcommands and subcommand processors by showing you alternate ways of structuring your utility.

Overview

To write a utility, perform the following steps:

1. Using the `EDIT_FILE` utility, create a file to contain the procedure in which you define the utility. (For more information about the `EDIT_FILE` utility, see the *NOS/VE File Editor manual*.)
2. Write the utility as follows:
 - a. Create a procedure and, within the procedure, define the utility. (This step is described in detail in *Example: Defining a Utility* next in this chapter.)
 - b. Define the subcommand processors for the utility subcommands. (Examples of subcommand processors are shown in *Example: Subcommand Processors* later in this chapter.)
3. To make your utility easier to read, format the utility file using the `SCL` formatter (this step is optional):
 - a. Create a file and place the utility definition in the file. In this *utility definition file*, you must enter the utility names and terminators on a single line, for example:

```
name=(date_time_utility, dtu) terminator=(quit, qui)
```

- b. Format the utility file you created in step 2. When formatting a procedure containing a utility, you must specify a utility definition file similar to the one you created in the preceding step. For example, the following command formats the utility file `$LOCAL.SAMPLE_UTILITY`:

```
/format_scl_proc input=$local.sample_utility ..
../output=$user.utility_file ..
../utility_definition_file=$local.utility_def
```

The formatted utility file now resides in the permanent file `$USER.UTILITY_FILE`. (The `SCL` formatter and utility definition files are discussed later in this chapter.)

4. Place the utility in an object library. For example, assume your utility, its command tables, and its subcommand processors are all in the permanent file `$USER.UTILITY_FILE`. Use the `CREATE_OBJECT_LIBRARY` utility to generate an object library as follows:

```
/create_object_library  
COL/add_modules l=$user.utility_file  
COL/generate_library l=$user.utility_library  
COL/quit
```

The utility now resides in the object library `$USER.UTILITY_LIBRARY`. (Object libraries are described in detail in the NOS/VE Object Code Management manual.)

5. Add the object library to your command list. For example, the following command adds the object library `$USER.UTILITY_LIBRARY` to your command list:

```
/create_command_list_entry e=$user.utility_library
```

The utility can now be executed simply by entering its name. Command lists are discussed in chapter 5, SCL Command and Procedure Execution. The `CREATE_COMMAND_LIST_ENTRY` command is documented in the NOS/VE Commands and Functions manual.

Example: Defining a Utility

When writing a command utility, you define the following items:

- The name of the command utility.
- The utility's environment.
- The subcommands of the utility.
- The statements to be executed within the established utility.

For an example of how to define these elements, see the sample utility (figure 9-2) and the description that accompanies it.

The sample utility displays the current date and time in various formats. It includes the following subcommands:

```
CHANGE_FORMAT
DISPLAY_DATE
DISPLAY_TIME
QUIT
```

For a discussion of the processors for these subcommands, see Example: Subcommand Processors later in this chapter.

For an executable version of this sample utility, see the `UTILITY_EXAMPLE` entry in the online Examples manual.

```
① PROC date_time_utility, dtu (  
    output, o : file = $output  
    status    : var of status = $optional  
    )  
  
② UTILITY name=date_time_utility prompt ='dtu' ..  
    library=:nve.bob.dtl#library  
③ command (change_format, chaf) processor=ntp#change_format  
    command (display_date, disd) processor=ntp#display_date  
    command (display_time, dist) processor=ntp#display_time  
    command (quit, end, qui) processor=ntp#quit  
④ tablend  
  
⑤ create_variable dtv#output scope=date_time_utility ..  
    kind=string value=$string($value(output))  
    create_variable dtv#date_format scope=date_time_utility ..  
    kind=string value='MONTH'  
    create_variable dtv#time_format scope=date_time_utility ..  
    kind=string value='AMPM'  
  
    include_file $command_of_caller u=date_time_utility  
  
⑥ UTILITYEND  
  
⑦ PROCEND date_time_utility
```

Figure 9-2. Sample SCL Command Utility

- ① You must define an SCL command utility within an SCL procedure. In the sample, the procedure header (PROC statement) identifies the procedure name `DATE_TIME_UTILITY` and its alias `DTU`. A call to the procedure begins execution of the enclosed utility.

Creating procedures is discussed earlier in this chapter.

- ② To begin the definition of the command utility, use the `UTILITY` command. The parameters of this command define the utility environment, for example:

- Use the `NAME` parameter to specify the name used to refer to the utility *while it is active* (`DATE_TIME_UTILITY`). This parameter is required.
- Use the `PROMPT` parameter to specify the prompt (`DTU`) that informs users they are in the utility. The following example is a call to the sample utility and the subsequent prompt display:

```
/date_time_utility
dtu/
```

- Use the `LIBRARY` parameter to specify the object library containing the processors for the utility subcommands (processors are discussed in the next step).

More complete descriptions of the `UTILITY` command and related commands are documented in the `NOS/VE Commands and Functions` manual.

- ③ To define a subcommand for the utility, use the **COMMAND** command. Each subcommand you define is an entry in a *command table* and can be used only while the utility is active.

You must define a subcommand that terminates the utility. An example of a termination subcommand is shown in Example: Subcommand Processors later in this chapter.

In the sample utility, the parameters of this command define:

- The names of the subcommand. These names are used to call the subcommand. The same rules apply to the subcommands you define as apply for listing procedure names and command names.
- The processor that executes when the subcommand is called. The processors listed in the sample utility are procedures residing on the object library :NVE.BOB.DTL#LIBRARY (defined by the **LIBRARY** parameter of the **UTILITY** command).

If you do not specify a processor for a subcommand, the first name in the command table entry is assumed to be the processor name as well. For example, the processor for the following command table entry:

```
command name=(my_subcommand,ms)
```

is assumed to be a procedure or program named **MY_SUBCOMMAND**.

- ④ To end the collection of entries for a utility's command table, use the **TABLEND** command. This command is required to separate command table entries from the executable statements in the established utility.

- ⑤ Enter the statements to be executed within the established utility. In the sample utility:
 - Three variables are defined to pass format and output information between the command processors. To see how these variables are used, see Example: Subcommand Processors later in this chapter.
 - The `INCLUDE_FILE` command associates `$COMMAND_OF_CALLER` with the utility and is necessary to enable interactive input within the utility. `INCLUDE_FILE`, in this case, accepts command input following the call to the utility (the `$COMMAND_OF_CALLER`) and passes the input to the utility for execution.
- ⑥ To end the definition of the command utility, use the `UTILITYEND` command. When the user executes the subcommand that terminates the utility, control is passed from the `UTILITY/UTILITYEND` block. This subcommand must be an entry in the command table you define for the utility.
- ⑦ Terminate the procedure with the `PROCEND` statement.

Example: Subcommand Processors

This section describes the procedures that act as subcommand processors for the sample utility in figure 9-2. For ease of reference, the definition of the utility block is repeated here.

```
UTILITY name=date_time_utility prompt='dtu' ..
    library=:nve.bob.dtl#library
    command (change_format, chaf) processor=ntp#change_format
    command (display_date, disd) processor=ntp#display_date
    command (display_time, dist) processor=ntp#display_time
    command (quit, end, qui) processor=ntp#quit
    tablend

    create_variable dtv#output scope=date_time_utility ..
        kind=string value=$string($value(output))
    create_variable dtv#date_format scope=date_time_utility ..
        kind=string value='MONTH'
    create_variable dtv#time_format scope=date_time_utility ..
        kind=string value='AMPM'

    include_file $command_of_caller u=date_time_utility

UTILITYEND
```

Processor for CHANGE_FORMAT

The following procedure acts as the processor for the CHANGE_FORMAT subcommand defined in the DATE_TIME_UTILITY. The subcommand has two parameters, DATE_FORMAT and TIME_FORMAT, which allow the user to specify the display formats for the utility's other subcommands.

If the user specifies a display format for the date, this value is passed to the utility block using the string variable DTV#DATE_FORMAT. If the user specifies a display format for the time, this value is passed to the utility block using the string variable DTV#TIME_FORMAT. If no display formats are specified, the default values defined in the utility block are used.

```
PROC dtp#change_format (
    date_format, df : key month, mdy, dmy, iso, ordinal = $optional
    time_format, tf : key ampm, hms = $optional
    status          : var of status = $optional
)

    create_variable dtv#date_format scope=xref kind=string
    create_variable dtv#time_format scope=xref kind=string

    IF $specified(date_format) THEN
        dtv#date_format = $string($value(date_format))
    IFEND
    IF $specified(time_format) THEN
        dtv#time_format = $string($value(time_format))
    IFEND

PROCEND dtp#change_format
```


Processor for DISPLAY_DATE

The following procedure acts as the processor for the DISPLAY_DATE subcommand defined in the DATE_TIME_UTILITY. The subcommand has two parameters, FORMAT and OUTPUT. The value for FORMAT is passed to the procedure from the utility block by the string variable DTV#DATE_FORMAT. The value for the OUTPUT parameter is passed to the procedure from the utility block by the string variable DTV#OUTPUT.

```
PROC dtp#display_date (  
    format, f : key month, mdy, dmy, iso, ordinal = ..  
        $name(dtv#date_format)  
    output, o : file = $fname(dtv#output)  
    status    : var of status = $optional  
    )  
  
    display_value $date($value(format)) o=$value(output)  
  
PROCEND dtp#display_date
```

Processor for DISPLAY_TIME

The following procedure acts as the processor for the DISPLAY_TIME subcommand defined in the DATE_TIME_UTILITY. The subcommand has two parameters, FORMAT and OUTPUT. The value for FORMAT is passed to the procedure from the utility block by the string variable DTV#TIME_FORMAT. The value for the OUTPUT parameter is passed to the procedure from the utility block by the string variable DTV#OUTPUT.

```
PROC dtp#display_time (
    format, f : key ampm, hms = $name(dtv#time_format)
    output, o : file = $fname(dtv#output)
    status    : var of status = $optional
)

    display_value $time($value(format)) o=$value(output)

PROCEND dtp#display_time
```

Processor for QUIT

The following procedure acts as the processor for the QUIT subcommand defined in the DATE_TIME_UTILITY. When the user specifies this subcommand, the utility block DATE_TIME_UTILITY is exited and control is passed to the enclosing procedure.

```
PROC dtp#quit (
)

    EXIT date_time_utility

PROCEND dtp#quit
```

Organizing Subcommands and Subcommand Processors

When you define the subcommands and subcommand processors for a utility by using the `COMMAND` command, you are creating a *command table*. This table identifies the names of the subcommands and subcommand processors.

When writing your own utilities, there are a number of ways you can organize the body of subcommands and subcommand processors. A discussion of some of these options follows.

Subcommands

When you define subcommands, you can place the definitions:

- In the utility block itself.
- In a separate file specified by the `TABLES` parameter on the `UTILITY` command.

In the sample utility in figure 9-2, the command table is defined within the `UTILITY/UTILITYEND` block. In this case, a `TABLEND` command is required to separate the entries in the command table from the executable statements in the established utility.

Alternately, you can specify a file containing the command table entries on the `TABLES` parameter of the `UTILITY` command. In this case, the `TABLEND` command is not required.

Using the `TABLES` parameter to specify the command list makes it easier to maintain the subcommands for a utility. You need not alter the utility file to update the utility's subcommands.

Subcommand Processors

Subcommand processors must be procedures or programs residing in an object library. When a subcommand is entered, the command list is searched for the name of the subcommand and its processor. The processor, if present, is then executed.

You can locate processors for the subcommands:

- In the same object library as the utility definition.
- In a separate object library specified by the `LIBRARY` parameter on the `UTILITY` command.

In the sample utility in figure 9-2, the processors are defined in a separate object library. If you do not specify an object library, the system assumes the processors are in the object library containing the utility.

Using the `LIBRARY` parameter to specify the object library in which a utility's subcommand processors reside makes it easier to maintain the subcommands for a utility. You need not alter the utility file to update the utility's subcommands.

Permitting Others to Use Your Utilities

You can permit other NOS/VE families and users to use the utilities you create. When you permit others to use your utility, you are actually permitting limited access to the object libraries and/or files you used to write your utility.

The structure of the utility file and the libraries and files associated with the utility determine the type of permit needed. The following two examples represent the configurations and permits you might use.

Example 1

The utility and its associated files reside in a single object library. That is:

- You defined the utility's command table in the `UTILITY/UTILITYEND` block.
- You placed both the utility definition file and the subcommand processors in a single object library.

When this is the case, use the `CREATE_FILE_PERMIT` command to permit access to the file. For example, to permit all NOS/VE users to use the utility residing in the object library `$USER.UTILITY_LIBRARY` enter:

```
/create_file_permit $user.utility_library ..  
../group=public access_mode=(read,execute) ..  
../share_mode=(read,execute)  
/
```

Any NOS/VE user can now add this library to their command list by specifying the complete file path name on the `CREATE_COMMAND_LIST_ENTRY` command.

For a more detailed description of file permits, see chapter 4, Catalog and File Management.

Example 2

The utility and associated files reside in more than one file and/or object library. That is:

- You specified a file to contain the command table using the `TABLE` parameter on the `UTILITY` command.
- You specified an object library in which the subcommand processors reside using the `LIBRARIES` parameter on the `UTILITY` command.

When this is the case:

1. Place the utility file and all associated files in a subcatalog of your `$USER` catalog.
2. Use the `CREATE_CATALOG_PERMIT` command to permit access to the files in the catalog.

For example, to permit users in the family `NVE` to use the utility residing in the subcatalog `$USER.UTILITY_CATALOG` enter:

```
/create_catalog_permit $user.utility_catalog ..
../group=family family_name=nve ..
../access_mode=(read,execute) ..
../share_mode=(read,execute)
/
```

Any user in the family `NVE` can now add the utility file to their command list by specifying the complete file path name on the `CREATE_COMMAND_LIST_ENTRY` command. The user need only add to the command list the `UTILITY/UTILITYEND` block. All associated files are accessible to the user with the catalog permit, provided that file references you made in the utility file give full path names.

For a more detailed description of catalog permits, see chapter 4, *Catalog and File Management*.

Using Your Utility

When using a utility you have defined, the same options are available to you as for any system-defined utility or SCL command:

- You execute the utility by name.
- You can be prompted for parameter input.
- You can display subcommand information.

You also have the option of changing some of the attributes of the utility with the `CHANGE_UTILITY_ATTRIBUTES` command.

Executing the Utility

Once you add the utility to your command list, you can execute it simply by entering its name (the name of the enclosing procedure). For example, to use the utility named `DATE_TIME_UTILITY` shown in figure 9-2:

1. Begin execution of the utility by entering its name.

```
/date_time_utility
```

2. When the utility prompt appears, enter the utility subcommands. For example:

```
dtu/display_time  
10:23 AM  
dtu/quit  
/
```

When you call a NOS/VE utility, it is typically executed as a separate task within your job. A task has the ability to establish an environment which is distinct from the environment which calls it.

One component of a task's environment is the command list, that is, the commands available for use within the task. Changes to the command list within a task are not necessarily recognized by other tasks within the same job.

For example, the `CREATE_OBJECT_LIBRARY` utility provides a set of subcommands for creating and maintaining object libraries. When the utility ends or the task is suspended by a call to another synchronous task, those subcommands are no longer available.

However, the utilities you create with the `UTILITY/UTILITYEND` statement are not executed in a separate task. Consequently, if you call the utility you create within a utility such as the `CREATE_OBJECT_LIBRARY` utility, you can use both utilities' subcommands.

Parameter Prompting

If you enter incorrect or incomplete parameter specifications during an interactive session, by default the system prompts you for further input. Parameter prompting also occurs by default for the subcommands you define for a utility.

For example, if you are using the `DATE_TIME_UTILITY` discussed earlier in this chapter, and you specify an invalid format on the `DATE_FORMAT` parameter for the `CHANGE_FORMAT` subcommand, the parameter prompting dialogue might appear as follows:

```
dtu/change_format date_format=year
Error: YEAR not an allowed value for parameter DATE_FORMAT.
Date Format? month
dtu/
```

For a more complete discussion of parameter prompting, see the Introduction to NOS/VE manual.

Displays of Subcommand Information

You can use the `DISPLAY_COMMAND_LIST_ENTRIES` command to display the subcommands of an active utility. For example, if you are using the `DATE_TIME_UTILITY` discussed earlier in this chapter and want a list of the utility subcommands, enter:

```
dtu/display_command_list_entries

ENTRY date_time_utility
Commands

change_format                display_date
display_time                 quit
dtu/
```

You can use the `DISPLAY_COMMAND_INFORMATION` command to display the parameters of a specific subcommand in an active utility. For example, if you are using the `DATE_TIME_UTILITY` discussed earlier in this chapter and want parameter information about the `CHANGE_FORMAT` subcommand, enter:

```
dtu/display_command_information change_format
date_format, df : key month, mdy, dmy, iso, ordinal = $optional
time_format, tf : key ampm, hms = $optional
status          : var of status = $optional
dtu/
```

The subcommand information is taken directly from the processors you define, is given a standard format, and is then displayed.

The `DISPLAY_COMMAND_LIST_ENTRY` and `DISPLAY_COMMAND_INFORMATION` commands are documented in the `NOS/VE Commands and Functions` manual.

Changing Utility Attributes

When using a utility you defined with the `UTILITY/UTILITYEND` command, you can change certain attributes of the utility using the `CHANGE_UTILITY_ATTRIBUTES` command.

The following example shows how you can change the utility attribute `ENABLE_SUBCOMMAND_LOGGING` (ESL) for the `DATE_TIME_UTILITY` discussed earlier in this chapter. (Because this parameter was not specified on the `UTILITY` command, the subcommands are logged by default.)

```

dtu/display_date
July 28, 1987
dtu/display_value $utility(subcommand_logging_enabled)
TRUE
dtu/change_utility_attributes utility=date_time_utility ..
dtu../enable_subcommand_logging=false
dtu/display_value $utility(subcommand_logging_enabled)
FALSE
dtu/display_date
July 28, 1987
dtu/display_log do=5
19:12:47.967.CI.display_date
19:12:52.638.CI.display_value $utility(subcommand_logging_
enabled)
19:12:56.977.CI.change_utility_attributes utility=
date_time_utility enable_subcommand_logging=false
19:13:01.059.CI.display_value $utility(subcommand_logging_
enabled)
19:13:15.694.CI.display_log do=5

```

While subcommand logging is enabled, the subcommand `DISPLAY_DATE` is recorded in the the job log (at 19:12:47). When the utility's `ENABLE_SUBCOMMAND_LOGGING` attribute is subsequently set to `FALSE`, the subcommand `DISPLAY_DATE` is not recorded in the job log.

The `CHANGE_UTILITY_ATTRIBUTES` command is documented in the `NOS/VE Commands and Functions` manual.

Formatting SCL Procedures

SCL provides a formatter that performs two major tasks:

- Formats a file containing SCL statements, making the procedure easier to read.
- Detects certain errors in the input file and issues appropriate diagnostic messages.

The formatter reads an input file of SCL statements and generates a formatted output file. The input file can consist of one or more SCL procedures or portions of procedures.

Formatting Example

The figures on the following pages illustrate the effect that the SCL procedure formatter has on an unformatted procedure. Figure 9-3 shows an unformatted procedure contained in file INFILE. Figure 9-4 shows the same procedure after formatting has taken place.

To format the sample procedure, perform the following steps:

1. Call the SCL procedure formatter as follows:

```
/format_scl_proc input=infile output=$user.proc_form
```

The procedure formatter processes the statements in input file INFILE and writes them to output file PROC_FORM in your \$USER catalog.

The FORMAT_SCL_PROC command is described in detail in the NOS/VE Commands and Functions manual.

Formatting Example

2. Enter the following command to view the formatted procedure (displayed in figure 9-4):

```
/copy_file file=$user.proc_form
```

```
proc aaa (  
input, i:file = $REQUIRED  
output, o:file=$OUTPUT  
status)  
put_this_message_out 'this is a message to be put to ..  
output' o=$output  
edit_file xyz  
l,,a  
include_file abcc  
quit  
while_label: while abc>def do  
if hij='???' then  
create_object_library  
add_module library=:nve.abcdefghijklmopqrstuvwxyz1.abcdefghi..  
jklmn2  
generate_library a  
quit  
else;exit  
ifend;while;procend
```

Figure 9-3. Unformatted SCL Procedure

```

PROC aaa (
  input, i : file = $required
  output, o: file = $OUTPUT
  status  : var of status = $optional
)

  put_this_message_out 'this is a message to be put to output' ..
    o=$output
  EDIT_FILE xyz
    1,, a
    include_file abcc
quit
while_label: ..
  WHILE abc > def DO
    IF hij = '???' THEN
      CREATE_OBJECT_LIBRARY
        add_module library=:nve.abcdefghijklmnpqrstuvwxy1..
        .abcdefghijklmn2
        generate_library a
    quit
  ELSE
    EXIT
  IFEND
  WHILEEND while_label

PROCEND aaa

```

Figure 9-4. Formatted SCL Procedure

Input to the Procedure Formatter

You create a file containing SCL statements for input to the procedure formatter. The file can consist of one or more SCL procedures, each beginning with a PROC statement and ending with a PROCEND statement.

It is not required that the input consist of complete procedures. Any collection of SCL statements is processed by the procedure formatter. However, SCL structure blocks within the input file must be complete, or the FORMAT_SCL_PROC command generates diagnostic messages.

Controlling the Formatting Process

To control the formatting process, you can enter special SCL comments in the input file. Begin a special SCL comment with the following characters:

"\$

Terminate a comment with either the double quote character (") or the end-of-line character.

Format The special comment has the following format:

```
"$  
  COMMAND=name  
  FORMAT=boolean
```

Parameters *COMMAND* (*C*)

Utility command or a utility terminator. This parameter informs the formatter that the specified command should be read by the formatter but ignored by the operating system when the procedure is executed.

FORMAT (*FMT* or *F*)

Boolean value that notifies the formatter whether to enable or disable the formatter. You can use this parameter to turn formatting on or off.

Remarks

- The comment can begin in any character position of a line, but only blank characters can precede the comment in the input line.
- If a command appears on the line after the special comment and is separated from it by a semicolon, the specified action applies to that command, as well as to following lines.

Examples Consider the following procedure fragment:

```
PROC show  
  :  
  edit_file xyz  
  include_file editing_directions  
  "$END  
  :  
PROCEND
```

The following activity occurs:

1. A call is made to the file editor command `EDIT_FILE`.
2. File `EDITING_DIRECTIONS` is executed.
3. The last editing subcommand in file `EDITING_DIRECTIONS` is `END`. To let the formatter know where the editor is terminated, the `"$END` comment is included in the procedure. When the procedure is executed, the `END` subcommand in the `EDITING_DIRECTIONS` file is executed and the `"$END` entry is ignored because it is a comment.

Utility Definition File

The SCL procedure formatter maintains a table of utilities and terminators. You can also add entries to this list.

Initial List of Utilities and Terminators

Table 9-3 lists the utilities initially entered into the utility table.

Table 9-3. Utilities and Terminators

Utility Name	Terminator
ADMINISTER_VALIDATIONS ADMV	QUIT QUI
ANALYZE_DUMP ANAD	QUIT QUI
BACKUP_PERMANENT_FILE BACKUP_PERMANENT_FILES BACPF	QUIT QUI
BUILD_REAL_MEMORY BUIRM	QUIT QUI
CREATE_APPLICATION_MENU CREAM	END_ APPLICATION_ MENU ENDAM QUIT QUI
CREATE_INTERSTATE_CONNECTION CREIC	DELETE_ INTERSTATE_ CONNECTION DELIC QUIT QUI
CREATE_MESSAGE_MODULE CREMM	END_MESSAGE_ MODULE ENDMM QUIT QUI

(Continued)

Table 9-3. Utilities and Terminators (Continued)

Utility Name	Terminator
CREATE_OBJECT_LIBRARY	QUIT
CREOL	QUI
DEFINE_CLIENT	END_DEFINE_
DEFC	CLIENT
	ENDDC
	QUIT
	QUI
DEFINE_SERVER	END_DEFINE_
DEFS	SERVER
	ENDDS
	QUIT
	QUI
DISPLAY_BINARY_LOG	QUIT
DISBL	QUI
EDIT_DECK	QUIT
EDID	QUI
	END
EDIT_FILE	QUIT
EDIF	QUI
	END
EDIT_PHYSICAL CONFIGURATION	QUIT
EDIPC	QUI
	END
LINK_VIRTUAL_ENVIRONMENT	QUIT
LINVE	QUI
LOGICAL_CONFIGURATION_UTILITY	QUIT
LOGCU	QUI
LCU	END
MAIL	QUIT
MAI	QUI

(Continued)

Table 9-3. Utilities and Terminators *(Continued)*

Utility Name	Terminator
MANAGE_NETWORK_APPLICATION	QUIT
MANAGE_NETWORK_APPLICATIONS	QUI
MANNA	
MEASURE_PROGRAM_EXECUTION	QUIT
MEAPE	QUI
NETWORK_OPERATOR_UTILITY	QUIT
NETOU	QUI
PHYSICAL_CONFIGURATION_UTILITY	QUIT
PHYCU	QUI
PCU	END
RESTORE_PERMANENT_FILE	QUIT
RESTORE_PERMANENT_FILES	QUI
RESPF	
SOURCE_CODE_UTILITY	QUIT
SCU	QUI
	END

Adding Utilities and Terminators to the List

To add to the list of utilities, complete the following steps:

1. Define a utility file.
2. Specify the utility file in the `FORMAT_SCL_PROC` command.

In the utility file, you must make entries on a *single* line that include the name of a utility and the utility terminator as follows:

```
NAME=list of name
TERMINATOR=list of name
```

NAME (NAMES or N)

List of names for one utility. This entry is required.

TERMINATOR (TERMINATORS or T)

List of names that terminate the utility. If you omit this entry, QUIT and QUI are used.

Utility names must be unique: they cannot be the same as the name of a system-defined or other utility. If the terminator names include more than QUIT and QUI, you must specify all terminators.

The following are examples of valid entries in a utility file:

```
n=(my_utility,mu) t=(end,quit,qui)
names=(your_utility, yu)
its_utility stop
```

How the Formatter Formats the Input File

The formatter generates lines in the output file according to the following conventions:

- Continuation lines:
 - Before an SCL statement is formatted, all continuation lines are read and concatenated with the first line of the statement.
 - If a statement line exceeds the specified page width, it is broken at a reasonable place, and required continuation lines are generated.
 - Continuation lines are indented six columns from the position at which the first line of the statement is indented.
 - When statement with continuation lines is too long to fit on the entire line if the indentation algorithm is obeyed, the indenting is suppressed in the continuation lines.
- Indent column:

The indent column governs the starting column of the statement written to the output file. Initially, the indent column is column 1 unless the INITIAL_INDENT_COLUMN parameter in the FORMAT_SCL_PROC command overrides.
- Indentations:

Following a control statement that defines the beginning of an SCL structure block, all statements are indented two spaces. Indentation ends with the corresponding end statement. (IF is an example of such an SCL control statement.)
- Labelled statements:

If a statement is labelled, the label (followed by the continuation ellipses) is placed on a line by itself. It is indented two columns less than the current indent column, if the current indent column is more than three.
- Uppercase/lowercase:

SCL control statement identifiers (IF, WHILE, PROC, etc.), along with certain control words (DO, EXIT, etc.) are written to the output file in uppercase. With the exception of certain utility names, all other names are written in lowercase.

- **Breaking expressions:**

When necessary, an expression consisting of an operand-operator-operand sequence is broken immediately after the operator.

- **Utilities:**

Utilities are formatted in the same manner as SCL structure blocks.

- **Comments:**

Comments occurring at the beginning of a statement line are written as separate lines and indented to the current indent column. Exceptions are comments starting in column 1 of the input file. Such comments are written starting in column 1 of the output file.

- **Spaces:**

With the exception of characters contained in a string expression or comment, spaces are added or deleted as follows:

- Commas are written with no leading space and with one trailing space.
- A colon that separates the parameter name from the parameter specification in a PROC declaration has leading spaces deleted and is followed by a space.
- Contiguous spaces are formatted as one space.
- A single space is inserted following the equal sign of an assignment statement.
- Unnecessary spaces in a parameter list are not written.

- **Multiple statements:**

If multiple statements separated by semicolons appear on an input line, each statement is formatted to a separate line and the semicolons are discarded.

- **Optional statement terminators:**

If a statement terminator is optional, the formatter generates the terminator when it is not specified on the input statement. (An example of an optional terminator is the THEN of an IF statement.)

- **COLLECT_TEXT and related commands:**

COLLECT_TEXT is treated as a special case, because the text being collected might not contain SCL statements. Whenever SCL encounters a COLLECT_TEXT command, the command's parameter list is scanned for the UNTIL parameter.

- If the UNTIL parameter is not specified, or if it is specified as a literal string, the formatting process stops until the line containing the specified string (or **) is read. All COLLECT_TEXT lines are copied to the output file without any other processing.
- If the UNTIL parameter is specified as a variable other than a literal string on COLLECT_TEXT, formatting does not stop, because there is no way of knowing when it is to be reactivated. The results in this case are undefined.

The COLLECT_TEXT command is converted to uppercase and written to the output file starting in column 1.

If you specify the PROCESS_COLLECT_TEXT parameter as TRUE, the COLLECT_TEXT lines are formatted.

The following commands are treated the same way as the COLLECT_TEXT command:

```
CREATE_BRIEF_HELP_MESSAGE  
CREATE_FULL_HELP_MESSAGE  
CREATE_PARAMETER_ASSIST_MESSAGE  
CREATE_PARAMETER_HELP_MESSAGE  
CREATE_PARAMETER_PROMPT_MESSAGE  
CREATE_STATUS_MESSAGE  
MANAGE_REMOTE_FILES
```

- **Key character:**

Any input line beginning with the key character specified in the `FORMAT_SCL_PROC` command is written directly to the output file with no attempt at processing.

- **Long statements:**

Statements that exceed the maximum command size of 65,535 characters are written to the output file as they appear in the input file. A warning message written as a comment is sent to the output file. However, the procedure formatter attempts to reduce the formatted statement to the maximum command size by decreasing any indentation.

- **Blank lines:**

If a blank line does not follow the last line of a procedure header, a blank line is added. If a blank line does not appear before a `PROCEND` statement, one is added.

How the Formatter Handles Errors

When the formatter encounters errors in the input file, the following conventions apply:

- If the formatter encounters a problem with a statement, such as the inability to determine the UNTIL value of the COLLECT_TEXT command, a warning message is written to the output file as a comment.
- Certain errors detected by the formatter (such as unbalanced parentheses) are reported to the output file as system diagnostic messages. The current message mode is used.
- For each error or warning message written to the output file, the input lines in which the error was detected immediately follow the message.
- Certain information about a line in error may be retained by the formatter. For example, if a WHEN statement is recognized but an error is detected in some remaining portion of the statement, a WHEN block is still established, and it requires a WHENEND statement.
- The formatter checks whether the block structures are properly nested. When it encounters a statement specifying the end of a block (such as WHILEND), the formatter checks for the definition of the WHILE block. If none exists and another block was defined within the block terminated by a WHILEND statement, a format error message is generated. The message indicates the line number of the definition of the statement that begins the internal block. When a PROCEND statement is encountered and a PROC block exists, all blocks begun but not terminated within the PROC block are noted in a diagnostic message.
- The formatter does not check parameter syntax. For example, the following statement is considered acceptable:

```
IF 'this is a string' = 255(10) THEN
```
- The formatter maintains a count of error and warning messages issued and reports this count to the status variable at termination of procedure formatting. However, an error count of 25 causes the formatting process to terminate.

Dual-State File Access

10

File Access Requirements	10-1
Getting a File from a Dual-State Partner System	10-2
Replacing a File on a Dual-State Partner System	10-2
Setting Link Attributes Using ADMINISTER_VALIDATION	10-3
Preserving NOS/VE File Attributes	10-4

A CYBER simultaneously running two operating systems is called a dual-state system. A NOS dual-state system runs NOS and NOS/VE; a NOS/BE dual-state system runs NOS/BE and NOS/VE. The pairs of operating systems on a dual-state system are called dual-state partners.

This chapter discusses copying files between NOS/VE and its dual-state partner. If your site is a dual-state site, you can print NOS/VE files on a printer belonging to the dual-state partner by using the `OUTPUT_DESTINATION_USAGE` and `REMOTE_HOST_DIRECTIVE` parameters of the `PRINT_FILE` command.

File Access Requirements

You can copy only files for which you are permitted access:

- If you are a NOS/VE user on a NOS dual-state system, you are typically validated to use both NOS and NOS/VE. Your user names for the two operating systems are the same, and you can access files in your permanent file catalog on either system. You can also access other NOS users' files if you have been permitted to access the files. (NOS controls access to files residing on the NOS system.) However, before you can access a file on NOS, you may have to use the `CHANGE_LINK_ATTRIBUTES` command to specify your NOS validation information.
- If you are a NOS/VE user on a NOS/BE dual-state system, you are typically validated to use both NOS/BE and NOS/VE. To access a file on NOS/BE, you must provide the correct NOS/BE file id and password, and the file must be on the NOS/BE default permanent file set. You can use the `CHANGE_LINK_ATTRIBUTES` command to provide alternate user and accounting information for the NOS/BE system.

If you use `CHANGE_LINK_ATTRIBUTES` to set your link attributes, you will typically need to use this command at least once for every job in which you are performing dual-state file accesses. To avoid this, you can set your link attributes once using the `ADMINISTER_VALIDATION` utility. See `Setting Link Attributes Using ADMINISTER_VALIDATION` later in this chapter for more information.

Getting a File from a Dual-State Partner System

To get a file from a dual-state partner system (NOS or NOS/BE) and copy it to a NOS/VE file, follow these steps:

1. Determine whether you need to use the `CHANGE_LINK_ATTRIBUTES` command to provide validation information to the partner system. You can determine what your current link attributes are by using the `DISPLAY_LINK_ATTRIBUTES` command. Your site may restrict the displaying of certain link attributes.
2. Establish the file attributes of the NOS/VE file to which you are going to copy the NOS or NOS/BE file:
 - If you are copying to a new NOS/VE file, you can use the `SET_FILE_ATTRIBUTE` command to set up the attributes of the file before you open it. Then you can reference the new file on the `GET_FILE` command.
 - If you are copying to an existing NOS/VE file, the attributes of the file stay the same.
3. Use the `GET_FILE` command to copy the file from NOS or NOS/BE to the NOS/VE file.

Replacing a File on a Dual-State Partner System

To copy a NOS/VE file to a file on a dual-state partner system (NOS or NOS/BE), follow these steps:

1. Determine whether you need to use the `CHANGE_LINK_ATTRIBUTES` command to provide validation information to the partner system. You can determine what your current link attributes are by using the `DISPLAY_LINK_ATTRIBUTES` command. Your site may restrict the displaying of certain link attributes.
2. Use the `REPLACE_FILE` command to copy the file to NOS or NOS/BE.

Setting Link Attributes Using ADMINISTER_VALIDATION

You can set your link attributes using the ADMINISTER_VALIDATION utility so that you do not have to specify a CHANGE_LINK_ATTRIBUTES command for every job in which you want to perform dual-state file accesses. Once you have set up a link attribute using the ADMINISTER_VALIDATION utility, that value will be the default for that attribute until you change it again using the utility.

In order to set link attributes using this utility, first enter the ADMINISTER_VALIDATION utility and then the CHANGE_USER subcommand. You can then use the following commands to set your link attributes:

CHANGE_LINK_ATTRIBUTE_CHARGE

Sets your charge number link attribute.

CHANGE_LINK_ATTRIBUTE_FAMILY

Sets your family name link attribute.

CHANGE_LINK_ATTRIBUTE_PASSWORD

Sets your password link attribute.

CHANGE_LINK_ATTRIBUTE_PROJECT

Sets your project number link attribute.

CHANGE_LINK_ATTRIBUTE_USER

Sets your user name link attribute.

For example, in order to set your password link attribute, enter the following:

```
/administer_validation
ADMV/change_user
CHAU/change_link_attribute_password value='clouds'
CHAU/quit
ADMV/quit
/
```

Preserving NOS/VE File Attributes

When you transfer a file to/from the dual-state partner, the file's attributes and permissions are not saved. You can use the permanent file backup and restore utilities to preserve this information. To do this:

1. Use the `BACKUP_PERMANENT_FILE` utility to back up the file(s) you want to move to the dual-state partner.
2. Use `REPLACE_FILE` with `DATA_CONVERSION=B56` to replace the NOS/VE backup file on the dual-state partner.
3. Use `GET_FILE` with `DC=B56` to get the NOS/VE backup file from the dual-state partner.
4. Use the `RESTORE_PERMANENT_FILE` utility to restore the file(s) that were backed up in step 1.

The following example demonstrates how to transfer the file `$USER.MY_COMMAND_LIB` to/from the NOS file `BACKUP`.

1. Backup the file `$USER.MY_COMMAND_LIB` to the backup file `$LOCAL.BACKUP`.

```
/backup_permanent_file bf=$local.backup l=$output
PUB/backup_file f=$user.my_command_lib
PUB/quit
```

2. To transfer the backup file to NOS, enter:

```
/replace_file f=$local.backup dc=b56
```

3. To obtain the backup file from NOS, enter:

```
/get_file t=$local.backup dc=b56
```

4. To restore the file as `$USER.MY_COMMAND_LIB_COPY`, enter:

```
/restore_permanent_file l=$output
PUR/restore_file f=$user.my_command_lib bf=$local.backup ..
PUR../nfn=$user.my_command_lib_copy
PUR/quit
```

Reserving and Releasing Tape Units	11-1
Reserving Magnetic Tape Units	11-2
Releasing Magnetic Tape Units	11-3
Requesting Magnetic Tapes	11-4
Rewinding Tape Files	11-5
NOS/VE Labelled Tape Support	11-5
Labelled Tape Terms	11-6
Tape Label Information	11-6
ANSI Volume Information	11-7
VOL1 Tape Label	11-7
ANSI File Information	11-8
HDR1 Tape Label	11-8
HDR2 Tape Label	11-9
NOS/VE HDR2 Extensions	11-10
Multifile Positioning	11-12
Label Processing	11-13
HDR Labels	11-13
EOF Labels	11-13
Using Labelled Tapes	11-16
Initializing Labelled Tape Volumes	11-16
Writing Labelled Tapes	11-17
Reading Labelled Tapes	11-18
Reading Tape Files Not Written on NOS/VE	11-18
Summary of Tape Label Attributes	11-19
Default Tape Label Attributes	11-24
Using the CHANGE_TAPE_LABEL_ATTRIBUTES Command	11-26
Displaying Tape Label Attributes	11-27
Labelled Tape File Examples	11-28
Writing an ANSI File Set	11-28
Reading a File Set	11-30
Duplicating Labelled Tapes	11-31
Using Unlabelled Tape Files	11-32
Reading and Writing Unlabelled Tapes	11-32
Skipping Tape Marks	11-34

This chapter describes how to reserve, release, request, and position magnetic tape resources. This chapter also describes labelled and unlabelled tape files, and tells you how to use them.

Reserving and Releasing Tape Units

If a job needs more than one tape file opened at the same time, the job must reserve the tape units before opening any of the tape files. Once the need for multiple tape file access is satisfied, the resources should be released and made available to other jobs.

The reservation of a tape unit does not result in actual tape unit assignment. Instead, NOS/VE records the information and, when the tape files are opened, ensures that the tape units are available to continue processing. The job is suspended if the required number of tape units is already reserved or assigned to other jobs. In this manner, NOS/VE can anticipate and prevent situations that would result in one or more jobs becoming deadlocked because an insufficient number of tape units is available to them.

Tape units remain reserved for a job until they are specifically released or until the job terminates. You should release tape reservations as soon as the job no longer requires them. If you do not release them, they remain in effect until the job terminates. Releasing the resources enables other jobs to access the tape units, enables NOS/VE to manage its tape resources more effectively, and increases throughput for all users.

Reserving Magnetic Tape Units

The `RESERVE_RESOURCE` command informs NOS/VE of the number of tape units needed for job processing.

This command is necessary only if a job requires the concurrent use of more than one tape unit. The maximum number of tape units which can be requested must not exceed the total number available within the system. If not, an error status is returned and the command is terminated.

The `RESERVE_RESOURCE` command ensures that the number of tape units needed by the requesting job will not conflict with the number of units that all other jobs have reserved or have assigned to them. If the reservation causes a conflict, the requesting job is suspended until other jobs release their tape reservations.

The `RESERVE_RESOURCE` command must be executed prior to opening a file that resides on tape. Once this command has been issued within a job, additional attempts to reserve resources are denied until all previous resource reservations have been released.

You should issue this command immediately before opening a tape file. Premature use of this command can delay job advancement.

The following example reserves three 9-track magnetic tape units: one with 800 cpi and two with 6250 cpi:

```
/reserve_resources mt9$800=1 mt9$6250=2
```

You should also immediately release resource reservations when the resources are no longer needed by using the `RELEASE_RESOURCE` command.

Releasing Magnetic Tape Units

The `RELEASE_RESOURCE` command releases tape reservations previously established with the `RESERVE_RESOURCE` command.

Tape reservations may be released as soon as they are no longer required by a job. For example, if four 9-track tape units were reserved and it is no longer necessary to have more than two of them in use concurrently, then two can be released immediately and the remaining two released later.

Tape units assigned to a file are returned to the system when the file is returned by a job. Even though the tape units are no longer assigned, the job's resource reservations remain in effect until released with a `RELEASE_RESOURCE` command or when the job terminates. When tape reservations are released they become available for other jobs.

The following example releases reservations for three 9-track magnetic tape units: one with 1600 cpi and two with 6250 cpi:

```
/release_resource mt9$1600=1 mt9$6250=2
```

Requesting Magnetic Tapes

The `REQUEST_MAGNETIC_TAPE` command associates a specific file with a tape unit. It provides NOS/VE with information used to direct assignment of a file to magnetic tape. Actual device assignment, access, or operator communication for tape mounting does not occur until the file is opened for access within the job.

If this command is issued for a file which is currently associated with a different device class such as mass storage or terminal, or if the tape file is already open, an error status is returned and the command is terminated.

The following example associates a 9-track, 1600 cpi tape with the external VSN of X01234 to file `PAYROLL`:

```
/request_magnetic_tape file=payroll type=mt9$1600 ..  
../external_vsn='X01234'
```

The following example associates a 9-track, 6250 cpi multi-volume tape file with external VSNs Y4567, Y4568, and Y4569 to file `NEWPL`:

```
/request_magnetic_tape file=newpl external_vsn=('Y4567', ..  
../'Y4568','Y4569') type=mt9$6250
```

The following example associates a tape with file `NEWPL`. When the file is opened, the operator will be requested to mount a tape with an external VSN of Y789. The system will then be asked to verify that a tape with a recorded VSN of Z408 was assigned. This form of the `REQUEST_MAGNETIC_TAPE` command is useful in situations where the tape's recorded VSN may not uniquely identify the volume.

```
/request_magnetic_tape file=tape_file external_vsn='Y789' ..  
../recorded_vsn='Z408' type=mt9$6250
```

If you are going to use two or more tape units simultaneously, use the `RESERVE_RESOURCE` command before using any `REQUEST_MAGNETIC_TAPE` commands.

Rewinding Tape Files

The `REWIND_FILE` command positions a file to the beginning-of-information. You can use other SCL commands to rewind tape files by including a file position on a parameter that specifies a file reference (for example, `$LOCAL.TAPE.$BOI`).

When `REWIND_FILE` is issued for an unlabelled tape file, the file is positioned to the beginning of the first tape volume.

If `REWIND_FILE` is issued for an ANSI-labelled tape file, and the value of the file's `FILE_SET_POSITION` tape label attribute is `NEXT_FILE`, the `REWIND_FILE` command causes the next ANSI-labelled file accessed to be the same as the previously accessed file. For `FILE_SET_POSITION` tape label attribute values other than `NEXT_FILE`, this command has no effect.

Labelled and unlabelled tapes, and tape label attributes are described later in this chapter.

NOS/VE Labelled Tape Support

NOS/VE supports labelled tapes that correspond to the 1978 ANSI standard X3.27 and the December 1983 revision. Included are the ANSI levels 1 through level 4 capabilities.

In NOS/VE, labelled tapes are supported for tape files having the following block type and record type file attributes:

- The `BLOCK_TYPE` is `USER_SPECIFIED (US)` and the `RECORD_TYPE` is either `FIXED (F)`, `UNDEFINED (U)`, `ANSI_VARIABLE (D)`, or `ANSI_SPANNED (S)`.
- The `BLOCK_TYPE` is `SYSTEM_SPECIFIED (SS)` and the `RECORD_TYPE` is `UNDEFINED (U)` or `VARIABLE (V)`.

Labelled Tape Terms

As you read about labelled tapes, be aware of the following terms:

ANSI file

A single file residing in a file set that is preceded by an HDR label group (HDR1 and optional HDR2 through HDR9 labels) and followed by an EOF label group (EOF1 and optional EOF2 through EOF9 labels). An ANSI file can span tape volumes. In the descriptions of labelled tapes, all files are assumed to be ANSI files unless otherwise noted. In the sections that follow, the terms ANSI file, and ANSI labelled file and ANSI labelled tape file are used interchangeably.

Tape File

A NOS/VE temporary file identified by a path name and a collection of tape volumes. The same path name can be used to access any of the ANSI-files residing on the file set associated with the tape file.

File Set

One or more ANSI-files residing on one or more tape volumes. Each file in the set has the same file set identifier but a unique file identifier.

Tape Label Information

The following sections describe the parts of an ANSI labelled file header that contain tape volume and tape file information.

ANSI Volume Information

ANSI tape volume information is contained in the VOL1 label which is recorded at the beginning of each volume.

VOL1 Tape Label

The VOL1 tape label contains the following information used to identify a tape volume:

Volume Identifier

Specifies the identifier used to verify that the correct tape volume is being accessed. The volume identifier is specified with the RECORDED_VSN parameter of the REQUEST_MAGNETIC_TAPE command.

Owner Identifier

Specifies the owner of the tape volume. When the system initializes a tape volume, the specified owner identifier is recorded in the VOL1 label of the tape. Currently, the owner identifier is ignored by the system when accessing a labelled tape volume.

Volume Accessibility Code

Specifies a validation code that must be associated with the user to access the tape volume. When the system initializes a tape volume, the specified volume accessibility code is recorded in the VOL1 label of the tape. Currently, a volume accessibility code is not required to access a tape volume.

ANSI File Information

ANSI file identification information is recorded at the beginning of each ANSI-labelled tape file. This information is contained in three areas which together constitute an HDR label group.

- HDR1 Tape Label
- HDR2 Tape Label
- NOS/VE HDR2 Extensions

Detailed descriptions of these three areas follow.

HDR1 Tape Label

The ANSI HDR1 tape label contains the following information for identifying and controlling access to a labelled tape:

File Identifier

Specifies the label identifier for the file. It is used to differentiate between different ANSI files in a file set.

File Set Identifier

Specifies a unique identification for a set of files at a site.

File Section Number

Specifies the section number of this portion of the ANSI file. When an ANSI file spans tape volumes, the portion of the file on the first volume is assigned section number 1; the portion on the second volume is assigned section number 2, etc.

File Sequence Number

Specifies the numeric position of an ANSI file on a multiframe set. You can use this sequence number for random positioning to an ANSI file on a multiframe set.

Generation Number

Specifies a particular revision of the ANSI file that is defined by the file identifier.

Generation Version Number

Specifies the state of processing of the ANSI file that is identified by the file identifier and generation number. This number is used to identify which steps, in a multistep file creation process, the ANSI file has undergone.

Creation Date

Specifies the creation date of the ANSI file.

Expiration Date

Specifies the earliest date that the ANSI file may be overwritten and, implicitly, the date that any subsequent files in the multifile set may be overwritten. Currently, the system does not perform expiration date checking.

File Accessibility Code

Specifies the validation code that a user must have to access the ANSI file. When the system writes a new ANSI file, the accessibility code is placed in the HDR1 label of the file. Currently the system does not perform accessibility code checking.

System Code

Identifies the system that created the ANSI file. ANSI files created on NOS/VE have a value of 'NOS/VE V1.0 '.

HDR2 Tape Label

The ANSI HDR2 tape label contains information that describes the format of the file data. When accessing an existing ANSI file, NOS/VE uses values from the HDR2 label if they are present; otherwise, the file attributes specified for the tape file are used. When you create a new ANSI file, the following items are recorded in the HDR2 label:

Record Format

Specifies the following ANSI record format:

- FIXED (F)
- ANSI_VARIABLE (D)
- ANSI_SPANNED (S)
- blank (non-ANSI standard format)

This item corresponds to the record_type file attribute. Different record_type values can be used for each ANSI file in a multifile set.

Block Length

Specifies the maximum size (in characters) for all tape blocks on the file. This item corresponds to the `maximum_block_length` tape label attribute (see Summary of Tape Label Attributes later in this chapter for more information). Different values can be used for each ANSI file on a multifile set.

Record Length

Specifies the maximum size (in characters) for all records on the ANSI file. This item corresponds to the `maximum_record_length` tape label attribute (see Summary of Tape Label Attributes later in this chapter for more information). Different values can be used for each ANSI file in a multifile set.

Buffer Offset Length

Specifies the number of characters at the beginning of each tape block that will be ignored prior to the beginning of the first record. In the current version of NOS/VE, only files with a buffer offset length of zero are supported.

NOS/VE HDR2 Extensions

NOS/VE maintains additional information in the ANSI HDR2 label that describes the format of ANSI files written in internal NOS/VE formats (non-ANSI record formats).

When accessing an existing ANSI file, NOS/VE uses values from the HDR2 label if they are present; otherwise, the NOS/VE tape file attributes are used.

When a new ANSI file is created by NOS/VE, the following information is recorded in the HDR2 label section that is reserved for system use:

Block Type

Specifies the following NOS/VE block types:

USER_SPECIFIED (US)

SYSTEM_SPECIFIED (SS)

Record Type

Specifies the following NOS/VE record types:

UNDEFINED (U)

FIXED (F)

VARIABLE (V)

ANSI_VARIABLE (D)

ANSI_SPANNED (S)

Padding Character

Specifies the NOS/VE padding character for fixed records. The padding character field applies only to tape files having user-specified blocking and F records.

Character Set

Specifies the character set in which the labels will be recorded. All labels on a file set are recorded in the same character set. This field also specifies the character set used for data on the ANSI file if the value of the character conversion field is TRUE. The valid character sets are ASCII and EBCDIC. Once the character set value is specified and the ANSI file is accessed, the character set value cannot be changed.

Character Conversion

Specifies whether character conversion is performed. To read ANSI files written in the EBCDIC character set, you must specify a character set value of EBCDIC and a character conversion value of TRUE.

To write ANSI files with EBCDIC data on a tape volume with EBCDIC labels, you must specify a character set value of EBCDIC and a character conversion value of TRUE.

ANSI files that require character conversion as well as those that do not may both reside on the same file set.

Unless you specify otherwise, NOS/VE assumes that textual information is represented by the ASCII character set.

Multifile Positioning

When accessing a tape volume containing more than one file, you can position the tape volume to the file you want. The following file positioning options are available, and can be specified on the CHANGE_TAPE_LABEL_ATTRIBUTES command.

Position	Description
BEGINNING_ OF_SET	First ANSI file on the file set.
CURRENT_FILE	Last ANSI file accessed.
NEXT_FILE	ANSI file following the one last accessed.
FILE_ IDENTIFIER_ POSITION	ANSI file identified by the file identifier and generation number.
FILE_ SEQUENCE_ POSITION	ANSI file identified by file sequence number.
END_OF_SET	Point immediately following the last file on the file set.

Label Processing

HDR labels (for example, HDR1 or HDR2 labels described earlier in this section) are read and verified each time an existing ANSI file is opened.

In the subsequent sections that pertain to labels, the READ, WRITE, and READ/WRITE access modes have the following meanings:

- WRITE access means that SHORTEN or APPEND access were specified. MODIFY access is not meaningful for tape files.
- READ access means that READ access was specified but SHORTEN and APPEND were not.
- READ/WRITE access means that READ access was specified as were SHORTEN or APPEND.

HDR Labels

HDR labels are written for a new or existing ANSI file if a new or existing ANSI file is opened with READ/WRITE or WRITE access and either of the following is true:

- The REWRITE_LABELS parameter is set to TRUE. (For a description of the REWRITE_LABELS parameter, see the description of the Summary of Tape Label Attributes.)
- The tape is currently positioned at a point immediately following the last file in the file set.

EOF Labels

EOF labels (EOF1 and EOF2) are written when the file is closed when either of the following conditions are true:

- HDR labels were written when the file was opened.
- The preceding operation was an output operation.

Table 11-1 summarizes the conditions for which HDR labels are written for an ANSI file.

Table 11-1. Conditions for Writing HDR Labels

FILE_SET_POSITION Value	Access Condition	HDR Labels Written
BEGINNING_OF_SET	REWRITE_LABELS=FALSE under READ access	No
	REWRITE_LABELS=FALSE under READ/WRITE access	No
	REWRITE_LABELS=FALSE under WRITE access	No
	REWRITE_LABELS=TRUE under READ access	No
	REWRITE_LABELS=TRUE under READ/WRITE access	Yes
	REWRITE_LABELS=TRUE under WRITE access	Yes
CURRENT_FILE¹	REWRITE_LABELS=FALSE under READ access	No
	REWRITE_LABELS=FALSE under READ/WRITE access	No ²
	REWRITE_LABELS=FALSE under WRITE access	No ²

1. Included with the **CURRENT_FILE** value are the **NEXT_FILE**, **FILE_IDENTIFIER_POSITION**, and **FILE_SEQUENCE_POSITION** values for the **FILE_SET_POSITION** tape label attribute.

2. If the **FILE_SET_POSITION** parameter of the **CHANGE_TAPE_LABEL_ATTRIBUTES** command is set to **FILE_SEQUENCE_POSITION** and the **FILE_SEQUENCE_NUMBER** value is one greater than the last file on the set, a new file is written at the end of the set. If the **FILE_SET_POSITION** parameter of the **CHANGE_TAPE_LABEL_ATTRIBUTES** command is set to **NEXT_FILE** and the previous file accessed was the last file of the file set, a new file is written at the end of the file set.

(Continued)

Table 11-1. Conditions for Writing HDR Labels (Continued)

FILE_SET_POSITION Value	Access Condition	HDR Labels Written
CURRENT_FILE¹	REWRITE_LABELS=TRUE under READ access	No
	REWRITE_LABELS=TRUE under READ/WRITE access	Yes ²
	REWRITE_LABELS=TRUE under WRITE access	Yes ²
END_OF_SET	REWRITE_LABELS=FALSE under READ access	No (error)
	REWRITE_LABELS=FALSE under READ/WRITE access	Yes
	REWRITE_LABELS=FALSE under WRITE access	Yes
	REWRITE_LABELS=TRUE under READ access	No (error)
	REWRITE_LABELS=TRUE under READ/WRITE access	Yes
	REWRITE_LABELS=TRUE under WRITE access	Yes

1. Included with the **CURRENT_FILE** value are the **NEXT_FILE**, **FILE_IDENTIFIER_POSITION**, and **FILE_SEQUENCE_POSITION** values for the **FILE_SET_POSITION** tape label attribute.

2. If the **FILE_SET_POSITION** parameter of the **CHANGE_TAPE_LABEL_ATTRIBUTES** command is set to **FILE_SEQUENCE_POSITION** and the **FILE_SEQUENCE_NUMBER** value is one greater than the last file on the set, a new file is written at the end of the set. If the **FILE_SET_POSITION** parameter of the **CHANGE_TAPE_LABEL_ATTRIBUTES** command is set to **NEXT_FILE** and the previous file accessed was the last file of the file set, a new file is written at the end of the file set.

Using Labelled Tapes

This section discusses initializing, writing, and reading labelled tapes.

Initializing Labelled Tape Volumes

To initialize a tape volume, you must contact the system operator; only the system operator can initialize a labelled tape. For further information about handling labelled tapes, see the NOS/VE Operations manual.

A tape volume that has been initialized by the system operator has a label consisting of a VOL1 label followed by an empty ANSI file. The format of the label is:



* Denotes Tapemark

To rewrite the labels on the empty ANSI file, you must specify TRUE for the REWRITE_LABELS parameter on the CHANGE_TAPE_LABEL_ATTRIBUTES command. You must also specify TRUE for REWRITE_LABELS if you want to write on an initialized tape volume or on an ANSI file not created by NOS/VE.

Writing Labelled Tapes

You can write a labelled tape file by copying the file from mass storage to the tape device. Before you issue the `COPY_FILE` command to copy the file to the tape, use the `CHANGE_TAPE_LABEL_ATTRIBUTES` command to specify that you want the tape label written. If you are writing a file to a tape volume that contains a file set, you also need to use the file positioning parameters on the `CHANGE_TAPE_LABEL_ATTRIBUTES` command.

The following example copies a single permanent mass storage file with V records and system-specified blocking to an ANSI labelled tape file. The example begins with the `REQUEST_MAGNETIC_TAPE` command to associate the tape file with the tape device. The `SET_FILE_ATTRIBUTES` command is then used to set the following file attributes for the tape file before the `CHANGE_TAPE_LABEL_ATTRIBUTES` command is issued:

```
RECORD_TYPE
MAXIMUM_RECORD_LENGTH
BLOCK_TYPE
FILE_LABEL_TYPE
```

When the `COPY_FILE` command is issued, the complete ANSI file is written including the tape label header.

```
/request_magnetic_tape file=$local.tape type=mt9$6250 ..
../recorded_vsn='TA9327' ring=true
/set_file_attributes file=$local.tape ..
../file_label_type=labelled block_type=user_specified ..
../record_type=fixed maximum_record_length=140
/change_tape_label_attributes file=$local.tape ..
../rewrite_labels=true
/copy_file input=$user.data_set_10 output=$local.tape
```

Reading Labelled Tapes

You can read an ANSI file by copying the file from tape to mass storage. The following example copies a single ANSI tape file (\$LOCAL.TAPE) to \$LOCAL.PF_1. In this example, the SET_FILE_ATTRIBUTES command specifies the file attributes for the tape file. Since no SETFA command is issued for the mass storage file, it inherits the file attributes and label information that were set for the tape file when it was written.

```
/request_magnetic_tape file=$local.tape type=mt9$6250 ..  
../recorded_vsn='TA9327' ring=false  
/set_file_attributes file=$local.tape ..  
../file_label_type=labelled block_type=user_specified ..  
../maximum_record_length=140 record_type=fixed  
/copy_file input=$local.tape output=$local.pf_1
```

You can find examples which show the way to read and write multiple ANSI files that are associated with file sets later in this chapter.

Reading Tape Files Not Written on NOS/VE

The following is an example that copies a tape file not written on NOS/VE to mass storage.

```
request_magnetic_tape file=$local.tape ..  
    recorded_vsn='show' type=mt9$6250 ..  
    ring=false  
set_file_attributes file=$local.tape ..  
    block_type=user_specified ..  
    record_type=fixed ..  
    maximum_block_length=132*100 ..  
    maximum_record_length=132 ..  
    file_label_type=labelled  
change_tape_label_attributes file=$local.tape ..  
    file_set_position=beginning_of_set  
copy_file input=$local.tape output=$local.data
```

Summary of Tape Label Attributes

Tape label attributes control and define the ways a labelled tape file is processed by NOS/VE. Tape label attributes can be set using the `CHANGE_TAPE_LABEL_ATTRIBUTES` command. This section discusses these attributes and how to use `CHANGE_TAPE_LABEL_ATTRIBUTES`. See Default Tape Label Attributes later in this section for information on the defaults for the tape label attributes

The following list describes each of the tape label attributes:

`BLOCK_TYPE`

Specifies the NOS/VE block type to be used to access the file. Values can be:

- `SYSTEM_SPECIFIED (SS)`
- `USER_SPECIFIED (US)`

`BUFFER_OFFSET`

Specifies the number of characters at the beginning of the each tape block that will be ignored prior to the beginning of the first record. Currently, only files with a buffer offset of 0 is supported.

`CHARACTER_CONVERSION`

Specifies whether or not file data is to be converted to or from the character set specified by the `CHARACTER_SET` attribute. Values are:

`TRUE`

Specifies that the file data will be converted. During a `READ` operation, the file is converted from the character set specified in the `CHARACTER_SET` attribute to ASCII when it is read by NOS/VE. During a `WRITE` operation, the tape file is written in the character set specified by the `CHARACTER_SET` attribute.

`FALSE`

No conversion takes place.

CHARACTER_SET

Specifies the character set of the labels and file data on the tape. Values can be ASCII or EBCDIC. All labels on the tape will be accessed in the character set specified by this attribute.

CREATION_DATE

Specifies the creation date of the ANSI file. This date is specified in ISO format (yy-mm-dd).

EXPIRATION_DATE

Specifies the expiration date of the ANSI file and, implicitly, the expiration date of any subsequent ANSI files in the volume set. This expiration date is specified in the ISO format (yy-mm-dd). If the expiration date is less than or equal to the creation date, a zero is recorded in the ANSI label when the ANSI file is written. Currently, the system does not perform expiration date checking.

FILE_ACCESSIBILITY_CODE

Specifies the 1-character validation code string that must be associated with users accessing the ANSI file. When writing an ANSI file, the system records the specified value in the HDR1 label on the tape file. Currently, when reading an ANSI file, the system ignores this attribute.

FILE_IDENTIFIER

Specifies the 1- to 17-character label identifier string used to differentiate between ANSI files on a multifile set.

FILE_SEQUENCE_NUMBER

Specifies the numeric position of an ANSI file on a multifile set. Use this attribute to randomly position the tape to any ANSI file on a multifile set. Values can be any integer from 1 to 9999.

If a value is specified for the FILE_SET_POSITION attribute, this attribute must be assigned a value. If no value is specified for FILE_SET_POSITION, this attribute is ignored.

FILE_SET_IDENTIFIER

Specifies a unique 1- to 6-character string identification for a set of ANSI files within an installation.

The value specified for this attribute is used for all subsequent ANSI files written to the file set if this attribute is specified on subsequent **CHANGE_TAPE_LABEL_ATTRIBUTES** commands for the same magnetic tape file.

FILE_SET_POSITION

Specifies the position of the ANSI file on the set of ANSI files that reside on the associated set of tape volumes.

The tape volumes are specified on a **REQUEST_MAGNETIC_TAPE** command prior to the **CHANGE_TAPE_LABEL_ATTRIBUTES** command entry.

Valid values for this attribute are:

BEGINNING_OF_SET (BOS)

During a **READ** operation, this value specifies that the first ANSI file on the file set is accessed. During a **WRITE** operation, this value specifies that the ANSI file written is the first one on the file set.

CURRENT_FILE (CF)

During a **READ** operation, this value specifies that the current ANSI file is to be read. That is, the last file accessed will be read again. During a **WRITE** operation, this value specifies that the current file is written (the last file accessed will be rewritten).

NEXT_FILE (NF)

During a **READ** operation, this value specifies that the ANSI file following the file last accessed will be read. During a **WRITE** operation, this value specifies that the ANSI file to be written follows the file last accessed. If the tape is positioned at the beginning of the first volume of the file set, the first ANSI file on the file set is accessed.

FILE_IDENTIFIER_POSITION (FIP)

When reading, this value specifies that the ANSI file identified by the FILE_IDENTIFIER and GENERATION_NUMBER attributes is to be accessed. When writing, this value specifies that the ANSI file identified by these attributes is to be rewritten. The FILE_IDENTIFIER and GENERATION_NUMBER values of the new ANSI file will be the same as those values for the existing ANSI file.

FILE_SEQUENCE_POSITION (FSP)

During a READ or WRITE operation, this value specifies that the ANSI file identified by the FILE_SEQUENCE_NUMBER attribute is to be accessed.

END_OF_SET (EOS)

When the REWRITE_LABELS attribute is TRUE, this value specifies that the ANSI file is to be written after the last ANSI file on the file set. When the REWRITE_LABELS attribute is FALSE, this value will cause an error to be returned.

GENERATION_NUMBER

Specifies a specific revision of the ANSI file defined by the FILE_IDENTIFIER attribute. Value can be an integer from 1 to 9999.

GENERATION_VERSION_NUMBER

Specifies the state of processing of the file specified by the FILE_IDENTIFIER and GENERATION_NUMBER attributes. Values can be any integer from 0 to 99. This value is used to identify which steps, in a multistep file creation process, the file has undergone.

MAXIMUM_BLOCK_LENGTH

Specifies the NOS/VE maximum block length used to access the ANSI file. Values can be an integer from 1 to 2,147,483,615. However, to read or write tape blocks that exceed 4,128 bytes, your site must be configured to allow long tape blocks.

MAXIMUM_RECORD_LENGTH

Specifies the NOS/VE maximum record length used to access the ANSI file. Values can be an integer from 1 to 4,398,046,511,103.

PADDING_CHARACTER

Specifies the NOS/VE padding character used to pad records for ANSI fixed record type (RT=F).

RECORD_TYPE

Specifies the record type used to access the ANSI file. Values are:

- FIXED (F)
- UNDEFINED (U)
- VARIABLE (V)
- ANSI_VARIABLE (D)
- ANSI_SPANNED (S)

REWRITE_LABELS

Specifies whether the HDR label group will be rewritten when the ANSI file is opened for READ/WRITE or WRITE access. Values are:

TRUE

Specifies that the HDR label group will be rewritten when the ANSI file is opened for READ/WRITE or WRITE access. TRUE is required for writing a new file over an existing file. It is also required for writing a new file subsequent to reading an existing file (unless the last file on the file set was read).

FALSE

Specifies that the HDR label group will not be rewritten when the ANSI file is opened for READ, READ/WRITE, or WRITE access. Refer to table 11-1 for more information.

Default Tape Label Attributes

The following table contains the default values of tape label attributes:

Table 11-2. Default Tape Label Attributes

Attribute Field	Default Value
BLOCK_TYPE	See footnote ¹
BUFFER_OFFSET	0
CHARACTER_CONVERSION	See footnote ¹
CHARACTER_SET	See footnote ¹
CREATION_DATE	Today's date
EXPIRATION_DATE	'00000' (implies that the ANSI file has expired)
FILE_ACCESSIBILITY_CODE	' ' (space)
FILE_IDENTIFIER	Leftmost 17 characters of the path name. (For example, if the NOS/VE tape file path is \$LOCAL.EXPERIMENT_34_RESULTS, the resulting FILE_IDENTIFIER value is EXPERIMENT_34_RES.)

1. If the REWRITE_LABELS parameter of the CHANGE_TAPE_LABEL_ATTRIBUTES command is FALSE, the values of these attributes are taken from the HDR2 label, if they are present. If you omit these parameters on the CHANGE_TAPE_LABEL_ATTRIBUTES command, and the REWRITE_LABELS parameter on the command is TRUE, the previously defined values for these file attributes are used and are recorded in the HDR2 label.

(Continued)

Table 11-2. Default Tape Label Attributes (Continued)

Attribute Field	Default Value
FILE_SEQUENCE_NUMBER	1 for the first access. For subsequent accesses, the file sequence number equals the sequence number of the file last accessed plus one.
FILE_SET_IDENTIFIER	Volume identifier from the VOL1 label.
FILE_SET_POSITION	NEXT_FILE ¹
GENERATION_NUMBER	1
GENERATION_VERSION_NUMBER	0
MAXIMUM_BLOCK_LENGTH	See footnote ²
MAXIMUM_RECORD_LENGTH	See footnote ²
PADDING_CHARACTER	See footnote ²
RECORD_TYPE	See footnote ²
REWRITE_LABELS	FALSE

1. This setting causes the ANSI file immediately following the current file to be accessed. If the tape is positioned at the beginning of the first volume of the file set, the first ANSI file on the file set is accessed.

2. If the REWRITE_LABELS parameter of the CHANGE_TAPE_LABEL_ATTRIBUTES command is FALSE, the values of these attributes are taken from the HDR2 label, if they are present. If you omit these parameters on the CHANGE_TAPE_LABEL_ATTRIBUTES command, and the REWRITE_LABELS parameter on the command is TRUE, the previously defined values for these file attributes are used and are recorded in the HDR2 label.

Using the CHANGE_TAPE_LABEL_ATTRIBUTES Command

When you write an ANSI file you can use the CHANGE_TAPE_LABEL_ATTRIBUTES command to write or change file labels.

You can define only one file at a time on the CHANGE_TAPE_LABEL_ATTRIBUTES command. Subsequent CHANGE_TAPE_LABEL_ATTRIBUTES commands for the same ANSI tape file merge with previous CHANGE_TAPE_LABEL_ATTRIBUTES commands.

Before you can use the CHANGE_TAPE_LABEL_ATTRIBUTES command you must first associate the tape file to your job with a REQUEST_MAGNETIC_TAPE command.

When writing more than one ANSI file to tape, you can use CHANGE_TAPE_LABEL_ATTRIBUTES to specify different HDR1 and HDR2 label information for each ANSI file written. If you use this command, it must precede the writing of each ANSI file. File attributes and tape label attributes not specified on this command are taken from any corresponding attribute values already in place for the file.

Displaying Tape Label Attributes

To display the current tape label attributes defined for an ANSI tape file, use the `DISPLAY_TAPE_LABEL_ATTRIBUTES` command.

- If you specify a `DISPLAY_OPTIONS` value of `NEXT_FILE` on this command, the values for the next ANSI file to be accessed are displayed.
- If you specify a `DISPLAY_OPTIONS` value of `CURRENT_FILE`, the values for the most recently accessed ANSI file are displayed.
- If your job has not opened the file referenced on this command and you have not referenced it on a `CHANGE_TAPE_LABEL_ATTRIBUTES` command, the default tape label attribute values are displayed (table 11-2).

An error occurs if the `NOS/VE` file has not been assigned to your job by a `REQUEST_MAGNETIC_TAPE` command.

Labelled Tape File Examples

The following examples illustrate the use of NOS/VE commands and in particular the `CHANGE_TAPE_LABEL_ATTRIBUTES` command to access and manipulate ANSI tape files.

Writing an ANSI File Set

The following example copies two mass storage files to a tape volume as a part of a file set. It also uses `DISPLAY_TAPE_LABEL_ATTRIBUTES` to list in `$USER.DISTLA_FILE_SET_1` the tape label attributes for each written file. Note that for each tape file on the `CHANGE_TAPE_LABEL_ATTRIBUTES` commands, the same file set, 'SET1', is specified for the `FILE_SET_IDENTIFIER` parameters but a unique file identifier is specified for the `FILE_IDENTIFIER` parameters.

```
/request_magnetic_tape file=$local.tape type=mt9$6250 ..  
../recorded_vsn='TA3642' ring=true  
/set_file_attributes file=$local.tape file_label_type=labelled ..  
../block_type=user_specified record_type=fixed ..  
../maximum_record_length=140  
/change_tape_label_attributes file=$local.tape ..  
../file_identifier='file1' file_set_identifier='set1' ..  
../rewrite_labels=true  
/copy_file input=$user.pf1 output=$local.tape  
/display_tape_label_attributes file=$local.tape ..  
../output=$user.distla_file_set_1  
/change_tape_label_attributes file=$local.tape ..  
../file_identifier='file2' file_set_identifier='set1' ..  
../rewrite_labels=true  
/copy_file input=$user.pf2 output=$local.tape  
/display_tape_label_attributes file=$local.tape ..  
../output=$user.distla_file_set_1.$eoi  
/copy_file input=$user.distla_file_set_1
```

The contents of the tape labels for the ANSI files that were written are on \$USER.DISTLA_FILE_SET_1 and appear as follows:

```
/edit_file file=$user.distla_file_set_1

FILE :$local.tape
Block_type           : user_specified
Buffer_offset       : 0
Character_conversion : no
Character_set        : ascii
Creation_date        : unknown
Expiration_date      : unknown
File_accessibility_code : ' '
File_identifier      : 'FILE1'
File_sequence_number : 1
File_set_identifier  : 'SET1'
File_set_position    : next_file
Generation_number    : 1
Generation_version_number : 0
Maximum_block_length : 4128
Maximum_record_length : 140
Padding_character    : ' '
Record_type          : ansi_fixed
Rewrite_labels       : yes
FILE :$local.tape
Block_type           : user_specified
Buffer_offset       : 0
Character_conversion : no
Character_set        : ascii
Creation_date        : unknown
Expiration_date      : unknown
File_accessibility_code : ' '
File_identifier      : 'FILE2'
File_sequence_number : 2
File_set_identifier  : 'SET1'
File_set_position    : next_file
Generation_number    : 1
Generation_version_number : 0
Maximum_block_length : 4128
Maximum_record_length : 140
Padding_character    : ' '
Record_type          : ansi_fixed
Rewrite_labels       : yes
```

Reading a File Set

The following example reads the files in the file set that were written in the previous example. Note the different uses of the parameters on each of the `CHANGE_TAPE LABEL_ATTRIBUTES` commands. On the first usage of the command, when a value of `FILE_SEQUENCE_POSITION` is specified for the `FILE_SET_POSITION` parameter, a value must also be specified for the `FILE_SEQUENCE_NUMBER` parameter. On the second usage of the command, when the value `FILE_IDENTIFIER_POSITION` is specified for the `FILE_SET_POSITION` parameter, a value must also be specified for the `FILE_IDENTIFIER` parameter.

```
/request_magnetic_tape file=$local.tape type=mt9$6250 ..  
../ring=false recorded_vsn='TA3642'  
/change_tape_label_attributes file=$local.tape ..  
../file_set_position=file_sequence_position ..  
../file_sequence_number=1 rewrite_labels=false  
/copy_file input=$local.tape output$local.first_file  
/change_tape_label_attributes file=$local.tape ..  
../file_identifier='file2' ..  
../file_set_position=file_identifier_position ..  
../rewrite_labels=false  
/copy_file input=$local.tape output=$local.second_file
```

Duplicating Labelled Tapes

By reserving two tape units, you can copy files from one tape directly onto another. The following example copies a labelled tape file (evsn='first') to the first file on a second labelled tape volume (evsn='second'). This example can be used to duplicate permanent file backup tapes that are not multi-ANSI files.

```

/reserve_resource mt9$6250=2
/request_magnetic_tape file=$local.first type=mt9$6250 ..
../ring=false evsn='first' rvsn='ta123'
/set_file_attributes file=$local.first ..
../file_label_type=labelled block_type=system_specified ..
../record_type=variable
/change_tape_label_attributes file=$local.first ..
../file_set_position=beginning_of_set rewrite_labels=false ..
../block_type=system_specified record_type=variable

/request_magnetic_tape file=$local.second type=mt9$6250 ..
../ring=true evsn='second' rvsn='ta67'
/set_file_attributes file=$local.second ..
../file_label_type=labelled block_type=system_specified ..
../record_type=variable
/change_tape_label_attributes file=$local.second ..
../file_set_position=beginning_of_set rewrite_labels=true ..
../block_type=system_specified record_type=variable

/copy_file i=$local.first o=$local.second
/release_resources mt9$6250=2

```


Using Unlabelled Tape Files

This section discusses how to read and write unlabelled tapes, and how to skip tape marks on unlabelled tapes.

Reading and Writing Unlabelled Tapes

You can write an unlabelled tape file by copying the file from mass storage onto a tape volume. The following example copies `$USER.DATA_SET_10`, a permanent mass storage file with V records and system-specified blocking, to an unlabelled tape file. When the permanent file is written to tape, it will be written with U records and user-specified blocking.

```
/request_magnetic_tape file=$local.tape type=mt9$6250 ..  
../ring=yes external_vsn='XU0023'  
/set_file_attributes file=$local.tape ..  
../file_label_type=unlabelled block_type=user_specified ..  
../record_type=undefined  
/copy_file input=$user.data_set_10 output=$local.tape
```

You can read an unlabelled tape file by first copying the file from tape to mass storage. The simplest way to do this is to use the `REQUEST_MAGNETIC_TAPE` command to associate the file with the magnetic tape device class and then copy the file into a mass storage file using the `COPY_FILE` command. When no file attributes are set for the output file, it inherits the file attributes of the input file named on the `COPY_FILE` command. For example:

```
/request_magnetic_tape file=$local.tape type=mt9$6250 ..  
../external_vsn='UL0067' ring=no  
/copy_file input=$local.tape output=$local.data
```

When you copy a file from tape to mass storage, you can use the `SET_FILE_ATTRIBUTES` command to set file attributes for both the input and output files. The following contains the previous example but includes the use of the `SETFA` command to specify record types and block types for both the input and output files.

```
/request_magnetic_tape file=$local.tape type=mt9$6250 ..
../external_vsn='UL0067' ring=no
/set_file_attributes file=$local.tape ..
../file_label_type=unlabelled block_type=user_specified ..
../record_type=undefined
/set_file_attributes file=$local.data ..
../block_type=system_specified record_type=variable
/copy_file input=$local.tape output=$local.data
```

By reserving two tape units, you can copy files from one tape directly onto another. This example copies an unlabelled tape file to another unlabelled tape volume.

```
/reserve_resource mt9$6250=2
/request_magnetic_tape file=$local.first ..
../type=mt9$6250 ring=false evsn='first'
/set_file_attributes file=$local.first ..
../file_label_type=unlabelled ..
../block_type=user_specified record_type=undefined

/request_magnetic_tape file=$local.second type=mt9$6250 ..
../ring=true evsn='second'
/set_file_attributes file=$local.second ..
../file_label_type=unlabelled block_type=user_specified ..
../record_type=undefined

/copy_file i=$local.first o=$local.second
/release_resources mt9$6750=2
```

Skipping Tape Marks

The `SKIP_TAPE_MARK` command positions an unlabelled tape file in a backward or forward direction. This command is not valid for labelled tape files.

The file is positioned a specified number of tape marks from the current position or until a boundary condition is encountered.

The boundary condition for a forward skip is the end of the last volume in the list of volumes associated with the file. The boundary condition for a backward skip is the beginning of the current volume. Refer to the `REQUEST_MAGNETIC_TAPE` command description later in this chapter for further information.

If the file is not associated with a magnetic tape unit, the command is ignored.

For example:

```
/skip_tape_marks file=master count=3  
/skip_tape_marks my_file forward 5  
/skip_tape_mark file=new_master direction=backward
```

Overview	12-1
Uses of the Backup Utility	12-2
Access Rules	12-3
List File Information	12-4
Backup Operations	12-5
Starting and Stopping the Backup Utility	12-5
Backing Up to Disk	12-6
Backing Up to Tape Files	12-6
Labelled Backup Tapes	12-6
Multivolume Files	12-7
Multifile Backup Tapes	12-7
Backing Up to \$NULL	12-9
Backing Up Catalogs	12-10
Backing Up a Specific Catalog	12-10
Excluding a Specific Catalog	12-10
Deleting Catalogs and their Contents	12-11
Backing Up Files	12-12
Backing Up a Specific File	12-12
Excluding a Specific File	12-13
Specifying File Cycles to Be Included	12-13
Including Cycles Based on Their Size	12-14
Excluding the Highest Cycles	12-16
Deleting All Cycles of a File	12-16
Summary of Backup Utility Subcommands	12-17
Restore Operations	12-19
Starting and Stopping the Restore Utility	12-19
Restoring from Disk and Tapes	12-20
Displaying Information about a Backup File	12-21
Using the \$BACKUP_FILE Function	12-23
Restoring Catalogs	12-23
Restoring a Nonexisting Catalog	12-24
Restoring an Existing Catalog	12-25
Restoring Files	12-25
Restoring All Files	12-25
Restoring a Nonexisting File	12-26
Restoring Cycles of an Existing File	12-26
Summary of Restore Utility Subcommands	12-27

This chapter describes the utilities that back up and restore permanent files. The chapter includes:

- General concepts, including file access rules for the utilities, instructions for starting and stopping the utilities, instructions for specifying list information, and instructions for specifying tape labels.
- Examples that illustrate how to use the subcommands of the utilities.

For information about backing up and restoring an entire system, refer to the Site Performance and Maintenance manual, Volume 2.

Overview

You can use the `BACKUP_PERMANENT_FILES` utility to create a backup file containing copies of files and catalogs. A backup file can reside on either disk or tape. `BACPF` provides a set of subcommands that allow you to select the files and catalogs to be copied to the backup file. `BACPF` copies all catalog information, such as file descriptions, access control lists, and usage logs.

When you want to restore files and catalogs from a backup file, you use the `RESTORE_PERMANENT_FILES` utility. `RESPF` provides a set of subcommands that allows you to select the files and catalogs you want to restore to the system from the backup file.

The rest of this chapter discusses the `BACKUP_PERMANENT_FILES` and `RESTORE_PERMANENT_FILES` utilities in detail.

Uses of the Backup Utility

Most sites regularly back up to tape all of the permanent files that exist at the site. However, you may also want to perform backup operation for the following reasons:

- If you want to maintain your own set of backup tapes for your files and catalogs rather than relying on your site's regular permanent file backups.
- If you want to transfer many files, or even whole catalogs, to other users' master catalogs, systems, or sites. You can do this by backing up all files and catalogs to a single backup file and then transferring just that file.
- If you want to rename a catalog. You can do it by performing the following steps:
 1. Back up the catalog to a disk file.
 2. Restore the backed up catalog using the name you wish it to have.
 3. Delete the catalog you wish to rename.

Access Rules

Because the backup and restore utilities access permanent files, the file access rules that apply to permanent file commands also apply to the utilities.

- To back up a file, you must have **READ** access permission.
- The backup utility will not back up a file cycle that is already attached to the current job with access modes that do not include **READ**, or with share modes that include **APPEND**, **MODIFY**, or **SHORTEN** access.
- The backup utility will not back up a file cycle that is attached by another job with a share mode that does not include **READ**, or with an access mode that includes **APPEND**, **MODIFY**, or **SHORTEN** access.
- To delete a file, you must have **CONTROL** access permission for the file.
- To delete a catalog, you must be the owner of the catalog.
- To restore a file cycle, you must have **CYCLE** permission for the existing file; to restore the initial cycle of a file you must have cycle permission for the catalog in which the file will reside.
- To restore a file with a password, you must be the owner of the file.
- To restore a catalog that does not reside in your permanent catalog structure, you must be the owner of the catalog.

List File Information

Both the backup and restore utilities summarize the results of the utility operations. This information is written to the file designated on the LIST parameter of the BACKUP_PERMANENT_FILES and RESTORE_PERMANENT_FILES commands. By default, the system lists the following information for each permanent file cycle:

- Modification date.
- Modification time.
- Size of the file.

You can specify the information you want on the list file as follows:

- For backup operations, enter the SET_LIST_OPTION subcommand prior to entering any other backup utility subcommands.
- For restore operations, enter the SET_LIST_OPTION subcommand prior to entering any other restore utility subcommands.

The SET_LIST_OPTION subcommand does not affect the list file information that the DISPLAY_BACKUP_FILE subcommand produces.

The following example uses the SET_LIST_OPTION subcommand to specify that the listing should show all the file and cycle display options.

```
/respf list=restore_list  
PUR/set_list_options file_display_option=all ..  
PUR../cycle_display_options=all
```

Backup Operations

The `BACKUP_PERMANENT_FILES` utility allows you to select and copy files and catalogs to a backup file. Backup files may reside on either disk or tape. The default is to back up to disk. To back up to tape, you must use the `REQUEST_MAGNETIC_TAPE` command before you start the backup utility.

Starting and Stopping the Backup Utility

To start the backup utility, use the `BACKUP_PERMANENT_FILES` command. You must include the name of the file to which you want the backed up information copied. You can also specify a file for list information.

The following example starts the backup utility, names the backup file `BACKUP_FILE`, and specifies `BACKUP_LIST` as the list file:

```
/backup_permanent_file backup_file=backup_file list=backup_list
```

Once you have started the utility, the system displays the permanent file backup utility prompt:

```
PUB/
```

In response to the prompt, you can enter backup utility subcommands to direct backup operations (table 12-1 summarizes the backup utility subcommands).

In a typical backup session, you would first issue a series of include and exclude subcommands to narrow the selection criteria for catalogs and files you want to back up. You would then use the `BACKUP_CATALOG` and `BACKUP_FILE` subcommands to specify the catalogs and files you want to back up.

You can also use the group of delete subcommands with the `EXCLUDE_HIGHEST_CYCLE` subcommand to delete all but the highest cycles of files in a catalog. (For an example, refer to the *Excluding the Highest Cycles* section of this chapter.)

Once you have used the subcommands to perform your backup, you can stop the utility by entering the `QUIT` subcommand:

```
PUB/ quit
/
```

Backing Up to Disk

When you backup a file or catalog, by default the backup utility will store the backup file on disk rather than on tape. When you backup to disk, the backup file will appear in your current working catalog under the name you specified on the `BACKUP_PERMANENT_FILE` command.

Backing Up to Tape Files

To backup to tape, you must use the `REQUEST_MAGNETIC_TAPE` command prior to entering the backup utility.

You can back up files to either labelled or unlabelled tapes. By default, the backup utility assumes a tape type of labelled. To change the default label type, enter the `CHANGE_BACKUP_LABEL_TYPE` command prior to executing any backup or restore operations. To display the current label type, use the `DISPLAY_BACKUP_LABEL_TYPE` command.

When you restore files from a backup tape, the default tape label type does not need to match the label type of the backup file. For example, if the default label type is labelled and the backup tape file label type is unlabelled, the system console operator specifies during the tape assignment whether the restore operation continues.

Instead of using the backup and restore utilities for tape backups, you can use the `COPY_FILE` command to copy a mass storage backup file to a labelled or unlabelled tape or to copy a backup tape file to a mass storage file.

For information on retrieving backup files from tape, see the Restore Operations section later in this chapter.

Labelled Backup Tapes

The first time you want to use a tape volume for a labelled tape backup, you must have the system console operator blank label the tape.

You can specify file position `$EOI` to append information to a labelled tape backup file.

Multivolume Files

Each catalog or file cycle which is backed up to tape is preceded by header information that contains file identification information, including the version number and file size.

The backup file on the first tape volume begins with this header information, followed by the file contents, followed by the next file header, and so on.

A backup file may begin on one volume and end on the next volume. Thus, the first information on tapes after volume 1 may not be the header information but actual file data. When this occurs, the restore utility issues the following message:

```
UNABLE TO READ THE VERSION NUMBER.
```

If you receive this message for the first volume, it indicates an error and you should inform the site analyst. For other volumes, the message is informational only: the restore utility skips forward to either the first file header or the first complete file on the volume.

Multifile Backup Tapes

You can put more than one backup file on a tape volume if you are backing up catalogs and files to labelled tapes. To do this, use the `CHANGE_TAPE_LABEL_ATTRIBUTE` command to set the `FILE_SET_POSITION` tape label attribute to `NEXT_FILE`.

Note that `NEXT_FILE` is the default for this attribute. However, unless you explicitly set this attribute to `NEXT_FILE`, the backup utility will change it to `BEGINNING_OF_SET` and you will only be able to have one backup file on the tape volume.

Once you have set the `FILE_SET_POSITION` to `NEXT_FILE`, you create multiple backup files on the labelled tape by issuing one `BACKUP_PERMANENT_FILE` command for each tape file.

You can make each backup file on the tape easier to retrieve by using the `FILE_IDENTIFIER` parameter on the `CHANGE_TAPE_LABEL_ATTRIBUTE` command to identify each backup file on the tape. By default, this string is blank. However, once you have specified a value for this string each subsequent file on the tape will inherit that value unless you specify a different `FILE_IDENTIFIER` for a file.

The following example uses the `BACKUP_CATALOG` subcommand to place three different backup files on a labelled tape:

```
/request_magnetic_tape f=tape evsn='abc' r=true
/chatla f=tape fsp=nf
/bacpf bf=tape
PUB/bacc c=$user.cobol
PUB/quit
/chatla f=tape fi='fortran catalog'
/bacpf bf=tape
PUB/bacc c=$user.fortran
PUB/quit
/chatla f=tape fi='cybil catalog'
/bacpf bf=tape
PUB/bacc c=$user.cybil
PUB/quit
/
```

Notice that the second and third files on this tape have file identifiers associated with them. Also, note that once you have explicitly specified `NEXT_FILE` for the `FILE_SET_POSITION` file attribute, you do not have to specify it again (not even on subsequent `CHANGE_TAPE_LABEL_ATTRIBUTES` commands).

The `BACKUP_PERMANENT_FILES` command and the `BACKUP_CATALOG` subcommand are discussed later in this chapter. For information on retrieving backup files from tapes, see the `Restore Operations` section later in this chapter.

Backing Up to \$NULL

When you are performing backup operations, it is not always necessary to back up files and catalogs to disk or tape; you may also back up to \$NULL (or to any null device class). When you back up to \$NULL, none of the specified catalogs or files are actually backed up. Rather, by default, a backup to \$NULL generates a list of all the files existing in the specified catalogs. The list will exist in the list file specified on the call to the backup utility. Therefore, this is a convenient way for you to generate a listing of every file currently existing in a catalog and its subcatalogs.

Backing up to \$NULL is also a good way of finding out which of your files, if any, contain unreadable data. Since a normal backup to \$NULL leaves the actual file data untouched by the backup utility, you must tell the backup utility you want it to attempt to read all of the file data during backup operations. This is done by specifying READ_DATA on the NULL_BACKUP_FILE_OPTION parameter of the SET_BACKUP_OPTIONS subcommand. This should be done immediately upon entering a backup utility session in which you are backing up to \$NULL and you want to locate any files that contain unreadable data. For example:

```
/bacpf bf=$null l=$local.list
PUB/setbo nbfo=read_data
PUB/bacc c=$user
PUB/quit
/
```

You may then locate any files containing unreadable data by examining the backup utility list file (\$LOCAL.LIST, in the preceding example) for error messages. Note that the NULL_BACKUP_FILE_OPTION parameter of the SET_BACKUP_OPTIONS subcommand will have no effect on those backup utility sessions where you are actually backing up to disk or tape.

Backing Up Catalogs

You may use the backup utility to perform the following operations on catalogs:

- Back up specific catalogs
- Exclude specific catalogs from subsequent BACPF operations.
- Delete the files from a catalog.
- Delete an entire catalog from a catalog.

The following sections discuss these operations in detail.

Backing Up a Specific Catalog

To create a backup copy of the file cycles and catalogs in a specific catalog, use the `BACKUP_CATALOG` subcommand.

The following example backs up all of your files, subcatalogs, and files residing in all of your subcatalogs:

```
PUB/backup_catalog c=$user
```

Excluding a Specific Catalog

To exclude a specific catalog from subsequent backup and delete operations, use the `EXCLUDE_CATALOG` subcommand.

The following example excludes subcatalog `CATALOG_1` from the `BACKUP_CATALOG` operation:

```
/bacpf bf=backup_file  
PUB/excc c=$user.catalog_1  
PUB/bacc c=$user
```

You can back up an excluded catalog by explicitly entering its name on the `CATALOG` parameter of the `BACKUP_CATALOG` subcommand.

Deleting Catalogs and their Contents

To delete all of the files within a catalog and within its subcatalogs, use the `DELETE_CATALOG_CONTENT` subcommand. To delete an entire catalog, use the `INCLUDE_EMPTY_CATALOG` subcommand before using the `DELETE_CATALOG_CONTENT` subcommand.

The following example deletes just the files residing within catalog `CATALOG_1`, and its subcatalogs; it does not delete any subcatalogs that might exist in `CATALOG_1`.

```
/bacpf bf=backup_file
PUB/delcc c=$user.catalog_1
PUB/quit
```

To delete `CATALOG_1` completely, use the `INCLUDE_EMPTY_CATALOG` subcommand. This subcommand determines whether empty catalogs are deleted by subsequent `DELETE_CATALOG_CONTENT` subcommands.

Note that all other subcommands take precedence over the `INCLUDE_EMPTY_CATALOG` subcommand.

The following example deletes `CATALOG_1` entirely from the master catalog:

```
/bacpf bf=backup_file
PUB/incec
PUB/delcc c=$user.catalog_1
PUB/quit
```


Backing Up Files

You may use the backup utility to perform the following operations on files:

- Back up a specific file.
- Exclude a specific file from a backup operation.
- Backup files based on file cycles.
- Delete specific files.

The following sections discuss these operations in detail.

Backing Up a Specific File

To create a backup copy of a specific permanent file, use the `BACKUP_FILE` subcommand.

The following example backs up all cycles of file `DATA_FILE_0` in subcatalog `CATALOG_1` of the master catalog:

```
PUB/backup_file f=$user.catalog_1.data_file_0
```

To backup a single cycle of a file, specify the cycle number in the file reference. For instance:

```
PUB/backup_file f=$user.catalog_1.data_file_0.87
```

Excluding a Specific File

To exclude a specific file from subsequent backup and delete operations, use the `EXCLUDE_FILE` subcommand. This subcommand takes precedence over all `INCLUDE` subcommands.

The following example excludes all file cycles of `DATA_FILE_1` in subcatalog `DATA_CATALOG_1` from the backup operation:

```
pub/exclude_file f=$user.data_catalog_1.data_file_1
pub/backup_catalog c=$user.data_catalog_1
```

You can back up an excluded file by explicitly entering its name on the `BACKUP_FILE` subcommand, as follows:

```
pub/backup_file f=$user.data_catalog_1.data_file_1
```

Or you can explicitly back up the highest cycle of the file with the following subcommand:

```
pub/backup_file f=$user.data_catalog_1.data_file_1.$high
```

Specifying File Cycles to Be Included

To specify the file cycles to be included in subsequent backup and delete operations, use the `INCLUDE_CYCLE` subcommand.

By using the `INCLUDE_CYCLE` subcommand, you can perform a partial backup or deletion of permanent files. You specify the cycles to be included in the operation based on their creation date and time, last access date and time, last modification date and time, or expiration date.

The `EXCLUDE` subcommands take precedence over this subcommand.

The following example produces a partial backup composed of all file cycles that were modified on or after January 2, 1988:

```
pub/include_cycle selection_criteria=modified_after 1 2 1988
```

Use two `INCLUDE_CYCLE` subcommands to specify a window for backup and delete operations. For example:

```
PUB/include_cycle selection_criteria=modified_after 1 1 1987
PUB/include_cycle selection_criteria=modified_before 1 3 1987
```

These subcommands cause only those cycles modified on January 1, 1987 and January 2, 1987 to be included in backup and delete operations.

The following example backs up and deletes all your file cycles that have not been accessed since April 30, 1986:

```
PUB/incc sc=accessed_before may 1, 1986
PUB/bacc c=$user
PUB/delcc c=$user
PUB/quit
```

Including Cycles Based on Their Size

To include cycles based on their size, use the `INCLUDE_LARGE_CYCLE` or `INCLUDE_SMALL_CYCLE` subcommand.

The `INCLUDE_LARGE_CYCLE` subcommand specifies that subsequent backup and delete operations include only permanent file cycles whose size is greater than or equal to a specified number of bytes.

The `INCLUDE_SMALL_CYCLE` subcommand specifies that subsequent backup and delete operations include only permanent file cycles whose size is less than or equal to a specified number of bytes.

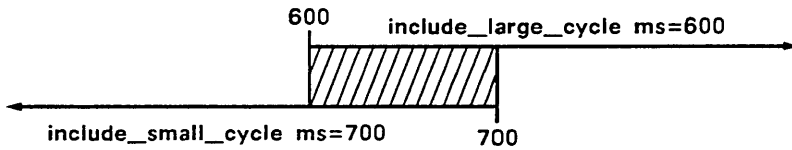
Regardless of its size, a file cycle that has been excluded using an `EXCLUDE` subcommand is not backed up or deleted.

Use these subcommands to reduce the size of the permanent file base by deleting large file cycles and ignoring small files cycles.

If you use both subcommands in a backup session, the intersection of the sets specified on the subcommands are included in subsequent backup operations. For instance, suppose you did the following in a backup session:

```
PUB/ include_large_cycle minimum_size=600
PUB/ include_small_cycle maximum_size=700
```

The following illustrates the size of the cycles included in subsequent backup operations (in this case, any cycles containing between 600 and 700 bytes):

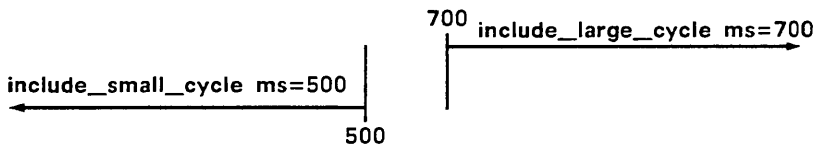


Indicates the size of the cycles included in subsequent backup operations.

Suppose in a different backup session, you did the following:

```
PUB/ include_large_cycle minimum_size=700
PUB/ include_small_cycle maximum_size=500
```

Then the following illustrates the size of the cycles included in subsequent backup operations (in this case, no cycles would be included because nothing is contained in the intersection of the sets specified by the two INCLUDE subcommands):



Excluding the Highest Cycles

To exclude the highest cycles from subsequent backup and delete operations, use the `EXCLUDE_HIGHEST_CYCLE` subcommand. (The highest cycles are the cycles with the largest numbers.) You specify the number of high cycles to be excluded.

The following example excludes the two highest cycles of each file in your master catalog from a subsequent `DELETE_CATALOG_CONTENT` subcommand:

```
PUB/exclude_highest_cycle noc=2
PUB/delcc c=$user
```

Deleting All Cycles of a File

To delete all cycles of a file, use the `DELETE_FILE_CONTENT` subcommand.

The following example deletes all cycles of permanent file `DATA_FILE_1` in your master catalog:

```
PUB/delfc f=$user.data_file_1
```

Summary of Backup Utility Subcommands

Table 12-1 summarizes the subcommands that direct the backup and delete operations.

Table 12-1. Summary of Backup Utility Subcommands

Subcommand	Purpose
BACKUP_CATALOG	Backs up the file cycles and subcatalogs in a catalog.
BACKUP_FILE	Backs up a permanent file cycle.
DELETE_CATALOG_CONTENTS	Deletes all files and subcatalogs in a catalog.
DELETE_FILE_CONTENTS	Deletes all cycles of a file.
EXCLUDE_CATALOG	Excludes a catalog from subsequent backup and delete operations.
EXCLUDE_FILE	Excludes a file or cycle from subsequent backup and delete operations.
EXCLUDE_HIGHEST_CYCLES	Excludes the highest cycles of files from subsequent backup and delete operations. (The highest cycles are the cycles with the largest numbers.)
INCLUDE_CYCLES	Determines the cycles included in subsequent backup and delete operations. You specify the cycles to be included based on the cycles' creation dates and times, last access dates and times, last modification dates and times, or expiration dates. An excluded cycle is not backed up or deleted, regardless of its date and time.

(Continued)

Table 12-1. Summary of Backup Utility Subcommands
(Continued)

Subcommand	Purpose
INCLUDE_EMPTY_CATALOGS	Causes subsequent DELETE_CATALOG_CONTENT subcommands to delete empty catalogs.
INCLUDE_LARGE_CYCLES	Causes subsequent backup and delete operations to include only cycles whose size is greater than or equal to the specified number of bytes. A previously excluded cycle is not backed up or deleted, regardless of its size.
INCLUDE_SMALL_CYCLE	Causes subsequent backup and delete operations to include only cycles whose size is less than or equal to the specified number of bytes. A previously excluded cycle is not backed up or deleted, regardless of its size.
SET_BACKUP_OPTIONS	Specifies subsequent actions to be taken by the backup utility.
SET_LIST_OPTIONS	Specifies the information you want subsequent subcommands to write to the list file.
QUIT	Terminates execution of the backup utility.

Restore Operations

The `RESTORE_PERMANENT_FILES` utility restores permanent files and catalogs from backup files created by the `BACKUP_PERMANENT_FILES` utility.

NOTE

The restore utility does not replace or overwrite file information that already exists on line.

Starting and Stopping the Restore Utility

To start the restore utility, use the `RESTORE_PERMANENT_FILE` command. On the command, you can specify a file for list information.

The following example starts the restore utility and specifies file `RESTORE_LIST` as the list file:

```
/respf l=restore_list
```

Once you have started the utility, the system displays the permanent file restore utility prompt:

```
PUR/
```

In response to the prompt, you can enter restore utility subcommands to direct restore operations (table 12-2 summarizes the restore subcommands).

Once you have used the subcommands to restore your files and catalogs, you can stop the utility by entering the `QUIT` subcommand:

```
PUR/quit  
/
```


Restoring from Disk and Tapes

By default, the restore utility assumes that a backup file exists on disk. Therefore, if the files and catalogs you are restoring exist in a backup file residing on tape, you must use the REQUEST_MAGNETIC_TAPE command before starting the restore utility.

To restore the contents of a backup file when more than one backup file resides on a labelled tape, use the CHANGE_TAPE_LABEL_ATTRIBUTE command to explicitly set the FILE_SET_POSITION tape label attribute to NEXT_FILE.

Unless you explicitly specify NEXT_FILE for this attribute, the restore utility will specify BEGINNING_OF_SET for the FILE_SET_POSITION tape label attribute.

For instance, suppose you have the following:

- A tape with an external volume serial number of ABC.
- Two backup files on the tape, each named TAPE.
- Each backup file contains one catalog. The first contains catalog \$USER.COBOL and the second contains catalog \$USER.FORTRAN.

To restore these catalogs, do the following:

```
/reqmt f=$local.tape evsn='abc' r=false
/chatla f=$local.tape fsp=nf
/respf
PUR/resc c=$user.cobol bf=$local.tape
PUR resc c=$user.fortran bf=$local.tape
PUR/quit
/
```

A backup file may also be placed on a labelled tape with a FILE_IDENTIFIER associated with it. You can then specify the backup file by entering a CHANGE_TAPE_LABEL_ATTRIBUTES command with the appropriate FILE_IDENTIFIER value and a FILE_SET_POSITION value of FILE_IDENTIFIER_POSITION.

For instance, suppose in the previous example that the second backup file on the tape had a file identifier associated with it of FORTRAN CATALOG. To restore the FORTRAN catalog from that file, you could do the following:

```
/reqmt f=$local.tape evsn='abc' r=false
/chatla f=$local.tape fsp=fip fi='FORTRAN CATALOG'
/respf
PUR/resc c=$user.fortran bf=$local.tape
PUR/quit
/
```

Displaying Information about a Backup File

To display information about the catalogs, files, and cycles of a backup file, use the DISPLAY_BACKUP_FILE subcommand.

You can select different informational displays with the DISPLAY_OPTION parameter. The following example displays the name and type of each entry on the backup file BACKUP:

```
/respf l=restore_list
PUR/display_backup_file bf=$local.backup ...
PUR./do=identifier
PUR/quit
/copf i=restore_list
DISPLAY BACKUP FILE:
:NVE.SARETT.BACKUP.1
-----
:NVE.SARETT.CATALOG_1
-----
:NVE.SARETT.CATALOG_1.DATA_FILE_0
PERMANENT FILE
-----
:NVE.SARETT.CATALOG_1.DATA_FILE_0
PERMANENT_FILE
CYCLE = 1
-----
:NVE.SARETT.CATALOG_1.DATA_FILE_0
PERMANENT_FILE
CYCLE = 2
-----
:NVE.SARETT.CATALOG_1.DATA_FILE_2
PERMANENT FILE
-----
:NVE.SARETT.CATALOG_1.DATA_FILE_2
PERMANENT_FILE
CYCLE = 1
```

To display more detailed information about each entry in the backup file, use the `DESCRIPTOR` display option on the `DISPLAY_BACKUP_FILE` subcommand. This information includes:

- Record headers maintained on the backup file.
- The version of the backup utility that produced the backup file.
- The date and time the backup file was written.
- The backup utility subcommand(s) that produced the backup file.
- Information on each file cycle that includes the following:
 - Cycle number.
 - Usage count.
 - Creation date and time.
 - Last access date and time.
 - Last modification date and time.
 - Expiration date.
 - Size in bytes.

If you specify the `READ_DATA` display option on the `DISBF` subcommand, the listing will report the following:

- All of the information returned for the `DESCRIPTOR` display option.
- The results of an attempt to read the data in every file cycle contained in the backup file.

Using the \$BACKUP_FILE Function

To display the attributes of a file or catalog on a backup file, use the `$BACKUP_FILE` function. This function returns the attribute information as a string with all letters converted to uppercase.

Because this function reads the backup file at the beginning-of-information, only the first item of information on the file can be queried and returned to you.

Use the `$BACKUP_FILE` function to restore a file or catalog when you do not know its name on the backup file, but do know the name of the destination file or catalog. For example, if you receive a tape produced by the `BACKUP_CATALOG` command and you want to restore a catalog on the backup file `BACKUP_FILE` to your own catalog, enter the following commands:

```
PUR/restore_catalog ..
PUR../c=$fname($backup_file(backup_file, identifier)) ..
PUR../backup_file=backup_file new_catalog_name=$user.my_catalog
```

The `$FNAME` function is included on the `RESTORE_CATALOG` subcommand to convert the string returned by `$BACKUP_FILE` to a file name. You can use the resulting file name in any subsequent `RESTORE_FILE` or `RESTORE_CATALOG` subcommand. These subcommands are discussed next in this chapter.

Restoring Catalogs

You can use the `RESTORE_PERMANENT_FILE` utility to restore catalogs from backup files. You can restore catalogs that do not currently reside within your permanent catalog structure, and you can restore catalogs that reside , but have some files or subcatalogs missing from them. The following sections discuss these two different types of catalog restoration.

Restoring a Nonexisting Catalog

To restore a nonresident catalog, use the `RESTORE_CATALOG` subcommand. The following example restores the master catalog to a new subcatalog in the master catalog:

```
PUR/restore_catalog
PUR../catalog=$user new_catalog_name=$user.catalog_2 ..
PUR../backup_file=backup_files
```

New subcatalog `CATALOG_2` appears in the master catalog as follows:

```
/display_catalog c=$user
CATALOG: CATALOG_1
CATALOG: CATALOG_2
  FILE: DATA_FILE_1
  FILE: EPILOG
  FILE: PROLOG
/display_catalog c=$user.catalog_2
CATALOG: CATALOG_1
  FILE: DATA_FILE_1
  FILE: EPILOG
  FILE: PROLOG
/display_catalog c=$user.catalog_2.catalog_1
  FILE: DATA_FILE_0
  FILE: DATA_FILE_2
/display_catalog_entry f=$user.catalog_2.catalog_1.data_file_0 ..
../display_option=descriptor
  NUMBER OF CYCLES:      2, ACCOUNT: D5927, PROJECT: P693N354
  PASSWORD: NEW_DATA_0_PW, LOG SELECTION: TRUE
  CYCLE NUMBER:      87, ACCESS COUNT: 3,
  CREATION DATE AND TIME: 1985-11-14 21:22:22.056,
  LAST ACCESS DATE AND TIME: 1985-11-14 21:23:24.531,
  LAST MODIFICATION DATE AND TIME: 1985-11-14 21:23:24.531,
  EXPIRATION DATE: 1985-01-13
  CYCLE NUMBER:      2, ACCESS COUNT: 2,
  CREATION DATE AND TIME: 1985-11-14 21:22:22.123,
  LAST ACCESS DATE AND TIME: 1985-11-14 21:23:24.531,
  LAST MODIFICATION DATE AND TIME: 1985-11-14 21:23:24.531,
  EXPIRATION DATE: NONE
```

Restoring an Existing Catalog

It is possible for you to have a catalog that is missing some or all of the data that it once contained. If you have previously backed up that catalog, you may restore its files and subcatalogs (and all files and subcatalogs within the subcatalogs) from the backup file, using the `RESTORE_EXISTING_CATALOG` subcommand. For example:

```
PUR/restore_existing_catalog ..
PUR../catalog=$user backup_file=backup_files
```

Note that since the restore utility never overwrites currently existing file cycles, this process will only restore data that exists in the backup file but not in the currently existing catalog.

Restoring Files

You can use the restore utility to perform the following operations on files:

- Restore all of the files that exist within the backup file and not in your catalog.
- Restore a specific file from the backup file.
- Restore the missing file cycles of an existing file.

The following sections discuss these operations in detail:

Restoring All Files

To restore all catalogs and permanent files on a backup file, use the `RESTORE_ALL_FILES` subcommand.

For example, the following job restores all files that were previously backed up during a backup utility session:

```
/job
job/request_magnetic_tape file=pf_tape_file ..
job../evsn='pfb001' type=mt9$6250 ring=false
job/respf
job/restore_all_files backup_file=pf_tape_file
job/quit
job/jobend
```

Restoring a Nonexisting File

To restore a file that does not reside in the permanent file system, use the `RESTORE_FILE` subcommand.

For example, the following commands restore cycle number 87 of file `DATA_FILE_0` in subcatalog `CATALOG_1`. The file cycle is restored as cycle number 1 of file `DATA_FILE_2` in `CATALOG_2` of the master catalog.

```
PUR/resf f=$user.catalog_1.data_file_0.87 ..  
PUR../bf=copy_of_file password=new_data_0_pw ..  
PUR../nfn=$user.catalog_2.data_file_2.1
```

Restoring Cycles of an Existing File

To restore missing cycles of a file that exists in the permanent file system, use the `RESTORE_EXISTING_FILE` subcommand.

The following example restores any cycles of file `$USER.CATALOG_1.DATA_FILE_0` that exist in backup file `COPY_OF_FILE` but that do not currently exist in the user's master catalog:

```
/delete_file f=$user.catalog_1.data_file_0.87 ..  
../pw=new_data_0_pw  
/respf  
PUR/restore_existing_file ..  
PUR../f=$user.catalog_1.data_file_0 ..  
PUR../backup_file=copy_of_file password=new_data_0_pw  
PUR/quit
```

Summary of Restore Utility Subcommands

Table 12-2 summarizes the subcommands that direct the restore utility operations.

Table 12-2. Summary of Restore Utility Subcommands

Subcommand	Purpose
<code>\$BACKUP_FILE</code> function	Returns attribute information for a file or catalog on a backup file produced by the backup utility.
<code>DISPLAY_BACKUP_FILE</code>	Displays the contents of a backup file produced by the backup utility.
<code>QUIT</code>	Terminates execution of the restore utility.
<code>RESTORE_ALL_FILES</code>	Restores the catalogs and permanent files in a backup file.
<code>RESTORE_CATALOG</code>	Restores a catalog that does not exist on line.
<code>RESTORE_EXISTING_CATALOG</code>	Restores the files and subcatalogs in a catalog.
<code>RESTORE_EXISTING_FILE</code>	Restores cycles of a file.
<code>RESTORE_FILE</code>	Restores file cycles that do not exist on line.
<code>SET_LIST_OPTION</code>	Specifies the information you want subsequent commands to write to the list file. The list produced by the <code>DISPLAY_BACKUP_FILE</code> subcommand is not affected.

Attribute Overview	13-1
Terminal Attributes	13-2
Connection Attributes	13-2
Network Dependencies	13-3
Attribute Value Constraints	13-3
Changing Attribute Values	13-3
Managing Terminal Attributes	13-4
Changing Terminal Attribute values	13-4
Displaying Terminal Attributes	13-4
Terminal Attribute Set	13-5
Terminal Attribute Defaults	13-17
Terminal Attribute Applicability	13-17
Network Type	13-17
Editing Mode	13-21
Managing Connection Attributes	13-23
Connection Attribute Levels	13-23
SCL Commands	13-24
Connection Attribute Set	13-25
Initial Default Values for Connection Attributes	13-34
Connection Attribute Applicability	13-36
Network Type	13-36
Editing Mode	13-38
Terminal Input	13-41
Input Buffering	13-41
Normal Editing Mode	13-42
Transparent Editing Mode	13-44
Typed-Ahead Input	13-45
Terminal Output	13-46
Normal Editing Mode	13-46
Interline Positioning	13-48
Page Holding	13-48
Transparent Editing Mode	13-49
Page Width and Page Length Attributes	13-50

This chapter discusses the attributes that define how the network and/or NOS/VE interacts with a terminal. It also describes the SCL and CDCNET commands that perform the following functions:

- Change and retrieve attribute values.
- Read data from a terminal or write data to a terminal.

An interactive job has only one terminal connection, and files that are assigned to the terminal device class within an interactive job are associated with the job's terminal connection.

An application can accept input from a terminal by reading a terminal file; an application can send output to a terminal by writing data to a terminal file. To associate a file with a terminal connection, you can use the REQUEST_TERMINAL command, described later in this chapter. For interactive jobs, the job files INPUT and OUTPUT are always assigned to the terminal device class.

Attribute Overview

The network maintains the attributes used for terminal management and determines their initial value. You can change or retrieve attribute values through either network commands or SCL commands. NOS/VE also changes certain attributes implicitly as part of performing I/O operations. (Refer to the Managing Connection Attributes section).

The attributes for terminal management are divided into two types: terminal attributes and connection attributes. Attribute usage for both types of attributes is network-dependent.

Terminal Attributes

The network maintains a single set of terminal attribute values for each terminal which apply to all connections from that terminal.

Terminal attributes describe physical characteristics of the terminal, which remain constant across all the connections from the terminal. For example, the attribute which specifies the parity used by the terminal is a terminal attribute. Terminal attributes are normally specified by the terminal user with either a network command or an SCL command.

Connection Attributes

Connection attributes describe how the terminal is used, which may vary from connection to connection. Thus, the network maintains a separate set of connection attribute values for each connection from a terminal. For example, the attribute which specifies whether characters received from the terminal are edited normally or transparently is a connection attribute. Connection attributes are normally specified by using an SCL command.

Network Dependencies

NOS/VE supports terminal access through the following networks:

- NAMVE/CDCNET
- NAM/CDCNET
- NAM/CCP
- INTERCOM

The connection attributes supported by NOS/VE correspond to those of the NAMVE/CDCNET network, along with several NAM/CCP attributes. Since the four networks do not offer identical capabilities, attribute usage depends on the specific network.

The attribute descriptions presented in this chapter specify the effect of the attributes for NAMVE/CDCNET connections. Some attributes may not be applicable, or may have a different effect for the other types of networks. Tables 13-1 and 13-4 indicate which attributes are supported by each type of network. The mapping from NOS/VE attributes to NAM/CCP device characteristics is described in the CYBIL File Management manual. Also included in the CYBIL manual are the attributes for NAM/CDCNET connections and the relationship of NOS/VE attributes to INTERCOM terminal support.

Attribute Value Constraints

Certain network-specific constraints exist on the values that may be assigned to attributes. For example, allowed values for one attribute may depend on the current value of another attribute. However, you do not usually encounter these constraints during typical use. For more information, refer to your network's documentation.

Changing Attribute Values

Both CDCNET and NAM/CCP provide network commands to change terminal and connection attributes. However, you should use SCL commands to change terminal and connection attribute values. You may use CDCNET network commands to change terminal attribute values, but you should not use them to change connection to change connection attribute values (see the Managing Connection Attributes section). You should not use NAM/CCP network commands to change the value of any attribute (device characteristic).

Managing Terminal Attributes

Terminal attribute values affect all connections from the terminal. In general, you should change terminal attributes only when you are beginning a terminal session.

Changing Terminal Attribute values

To change terminal attributes, use one of the following commands:

The SCL `CHANGE_TERMINAL_ATTRIBUTE` command.

The CDCNET `<ncc>CHANGE_TERMINAL_ATTRIBUTES` command where `<ncc>` is the CDCNET network command character (usually a `%`).

See the Terminal Attribute Set section for attributes, names, and descriptions.

Displaying Terminal Attributes

The `DISPLAY_TERMINAL_ATTRIBUTE` command displays the attribute values of the terminal in use for the job.

The following example lists the indicated terminal attributes:

```
/display_terminal_attributes ..  
../(cancel_line_character, ..  
../backspace_character)  
Backspace_Character           : $CHAR(8)      "BS"  
Cancel_Line_Character         : $CHAR(24)    "CAN"
```

Note that the quoted characters on the right are ASCII mnemonics.

Terminal Attribute Set

Terminal attributes describe the physical characteristics of a terminal and remain constant across all connections from the terminal. The following brief descriptions of each terminal attributes include:

- The attribute's parameter name and abbreviation.
- The attribute's purpose.
- Valid attribute values.

Further descriptions of the functions of terminal attributes can be found in the CDCNET Terminal Interface Usage manual.

ATTENTION_CHARACTER (AC)

Specifies the input character which causes the network to perform the action specified by the ATTENTION_CHARACTER_ACTION connection attribute. The ATTENTION_CHARACTER is recognized whenever it is received from the terminal (it does not need to occur at the beginning of a line). You can set this attribute to the NUL character to disable this feature.

BACKSPACE_CHARACTER (BC)

Specifies the input character which causes the network to delete the previous character in an input line. The effect of this attribute is conditioned by the value of the STORE_BACKSPACE_CHARACTER connection attribute.

BEGIN_LINE_CHARACTER (BLC)

Specifies the input character which the terminal sends at the beginning of a line. When the specified character is received as the first character of an input line, the network discards the character. This attribute is useful for block mode terminals, which send a fixed character at the beginning of a line and at the end of a line. You can set this attribute to the NUL character to disable this feature.

CANCEL_LINE_CHARACTER (CLC)

Specifies the input character that, when followed by the END_LINE_CHARACTER, causes the network to cancel the line being entered. The network forwards the cancelled line to NOS/VE which then discards the line and any partial input line that preceded it. You can set this attribute to the NUL character to disable this feature.

CARRIAGE_RETURN_DELAY (CRD)

Specifies the number of milliseconds the network is to wait before sending additional output to the terminal after a carriage return action has been performed. The delay allows a mechanical printing mechanism to reposition before printing is resumed. While the delay is active, NUL characters are sent to the terminal.

CARRIAGE_RETURN_SEQUENCE (CRS)

Specifies the 0- to 2-character string sent to the terminal to perform a carriage return action. See the description of the END_LINE_POSITIONING, and END_PARTIAL_POSITIONING terminal attributes for more information.

CHARACTER_FLOW_CONTROL (CFC)

Specifies whether the X-ON/X-OFF protocol (DC1 and DC3 characters) is to be used to regulate the flow of data between the network and the terminal.

TRUE

The X-ON/X-OFF protocol is to be used to regulate input and output.

FALSE

The X-ON/X-OFF protocol is not to be used.

CODE_SET (CS)

Specifies the character encoding used by the terminal. The following values are allowed:

ASCII

The terminal uses the ASCII character set.

TPAPL

The terminal uses the typewriter-paired APL character set.

BPAPL

The terminal uses the bit-paired APL character set.

ECHOPLEX (E)

Specifies whether each input character received from the terminal is sent back (echoed) to the terminal by the network.

TRUE

Input is echoed to the terminal.

FALSE

Input is not echoed to the terminal.

END_LINE_CHARACTER (ELC)

Specifies the input character that indicates the end of a complete input line. This character causes the network to forward the stored input characters to NOS/VE as a complete input line. The **END_LINE_CHARACTER** is not forwarded as part of the data.

END_LINE_POSITIONING (ELP)

Specifies the character string sent to the terminal to position the cursor upon receipt of the END_LINE_CHARACTER. The following values are allowed:

CRS

The character string sent is the value of the CARRIAGE_RETURN_SEQUENCE terminal attribute.

LFS

The character string sent is the value of the LINE_FEED_SEQUENCE terminal attribute.

CRSLFS

The character string sent is the value of the CARRIAGE_RETURN_SEQUENCE terminal attribute followed by the value of the LINE_FEED_SEQUENCE terminal attribute.

NONE

No character string is sent.

END_OUTPUT_SEQUENCE (EOS)

Specifies a string of 0 to 4 characters that is sent to the terminal following every complete output message. (Refer to the Terminal Output section for a description of a complete output message.) Any character combination is allowed. This attribute is provided for use by test tools such as terminal emulator packages.

END_PAGE_ACTION (EPA)

Specifies the character string sent to the terminal after a page of output data has been sent to the terminal without an intervening input line. This attribute has no effect if the PAGE_LENGTH terminal attribute is set to 0. If HOLD_PAGE is TRUE, the page holding action occurs before the character string is sent to the terminal. The following values are allowed:

FFS

The character string sent is the value of the FORM_FEED_SEQUENCE terminal attribute.

NONE

No character string is sent.

END_PARTIAL_CHARACTER (EPC)

Specifies the input character which indicates the end of a partial input line. This character causes the network to forward the stored input characters to NOS/VE as a partial input line. The **END_PARTIAL_CHARACTER** is not forwarded as part of the data. The effect of this attribute is conditioned by the value of the **PARTIAL_CHARACTER_FORWARDING** connection attribute. You can set this attribute to the NUL character to disable this feature.

END_PARTIAL_POSITIONING (EPP)

Specifies the character string sent to the terminal to position the cursor upon receipt of the **END_PARTIAL_CHARACTER**. The following are allowed values:

CRS

The character string sent is the value of the **CARRIAGE_RETURN_SEQUENCE** terminal attribute.

LFS

The character string sent is the value of the **LINE_FEED_SEQUENCE** terminal attribute.

CRSLFS

The character string sent is the value of the **CARRIAGE_RETURN_SEQUENCE** terminal attribute followed by the value of the **LINE_FEED_SEQUENCE** terminal attribute.

NONE

No character string is sent.

FOLD_LINE (FL)

Specifies whether the network folds output lines whose length exceeds the value of the PAGE_WIDTH terminal attribute.

TRUE

The network folds output lines when necessary. The portion of an output line that exceeds the PAGE_WIDTH value is displayed on the next physical line.

FALSE

The network does not fold output lines. If line folding is to occur, it must be performed by the terminal.

FORM_FEED_DELAY (FFD)

Specifies the number of milliseconds the network is to wait before sending additional output to the terminal after a form feed action has been performed. The delay allows a mechanical printing mechanism to reposition before printing is resumed. While the delay is active, NUL characters are sent to the terminal.

FORM_FEED_SEQUENCE (FFS)

Specifies the 0- to 7-character string sent to the terminal to perform a form feed action. See the description of the END_PAGE_ACTION terminal attribute for more information.

HOLD_PAGE (HP)

Specifies whether the network suspends the flow of data to the terminal when a page of output data has been sent to the terminal without an intervening input line. This attribute has no effect if the PAGE_LENGTH terminal attribute is set to 0. (Refer to the Terminal Output section for a description of page holding.)

TRUE

Terminal output is suspended when a page of output has been displayed.

FALSE

Output is sent to the terminal without interruption.

HOLD_PAGE_OVER (HPO)

Specifies whether the network sends a prompt to the terminal each time a hold page condition occurs. (Refer to the Terminal Output section for a description of page holding.)

TRUE

A prompt message is sent to the terminal after a page of output has been displayed.

FALSE

No prompt is sent to the terminal.

LINE_FEED_DELAY (LFD)

Specifies the number of milliseconds the network is to wait before sending additional output to the terminal after a line feed action has been performed. The delay allows a mechanical printing mechanism to reposition before printing is resumed. While the delay is active, NUL characters are sent to the terminal.

LINE_FEED_SEQUENCE (LFS)

Specifies the 0- to 2-character string sent to the terminal to perform a line feed action. See description of the **END_LINE_POSITIONING** and **END_PARTIAL_POSITIONING** terminal attributes for more information.

NETWORK_COMMAND_CHARACTER (NCC)

Specifies the character used to identify network commands. When this character is the first character of an input line, the line is processed by the network and not forwarded to NOS/VE.

PAGE_LENGTH (PL)

Specifies the number of lines displayed at the terminal as a page of output. When an end of page condition occurs (the page length value minus one physical line of data have been sent to the terminal), the network performs the action specified by the **HOLD_PAGE** terminal attribute and then the action specified by the **END_PAGE_ACTION** terminal attribute. A value of 0 indicates an infinite page length, meaning that an end of page condition did not occur.

PAGE_WIDTH (PW)

Specifies the number of characters that the terminal can display on a line. A value of 0 indicates an infinite page width, meaning that the network does not perform line folding. In order to allow NOS/VE programs to tailor output to the terminal's actual line width, you should set this attribute to indicate the physical characteristics of the terminal.

PARITY (P)

Specifies the parity checking performed on each character received from the terminal and the parity generation performed for each character sent to the terminal. The following values are allowed:

EVEN

The sum of all bits in a character is an even number.

ODD

The sum of all bits in a character is an odd number.

MARK

The parity bit is set to one.

NONE

If the INPUT_EDITING_MODE connection attribute is set to TRANSPARENT, no parity check is performed on input and no parity is generated on output. If the INPUT_EDITING_MODE connection attribute is NORMAL, no parity check is performed on input, but the parity bit is set to zero in each character sent to NOS/VE and each character sent to the terminal.

ZERO

The parity bit is set to zero.

PAUSE_BREAK_CHARACTER (PBC)

Specifies the input character that causes a pause break condition when it is received as the only character on a line.

STATUS_ACTION (SA)

Specifies how the network handles status messages from network operators. The following values are allowed:

DISCARD (D)

Status messages received by the network are not displayed at the terminal.

SEND (S)

Each status message is displayed at the terminal when it is received by the network.

HOLD (H)

The four most recent status messages are held by the network and not displayed until either the connection terminates or the value of this attribute is changed.

TERMINAL_CLASS (TC)

Specifies the class of terminal in use. The following values are allowed:

TTY	M3x teletypewriters.
C75x	CDC 75x, 722-10, 722-20 terminals.
C721	CDC 721 terminals.
I2741	IBM 2741 terminals.
TTY40	M40 teletypewriters.
H2000	Hazeltine 2000 terminals.
X364	ANSI X3.64 terminals, including CDC 722-30 terminals.
T4010	Tektronix 4010 terminals.
HASP_POST	HASP terminals that support only postprint format effectors.

HASP_PRE	HASP terminals that support both postprint and preprint format effectors.
C200UT	CDC 200 user terminals.
714_30_40	CDC 714-30 and CDC 714-40 terminals.
C711	CDC 711 terminals.
C714_10_20	CDC 714-10 and CDC 714-20 terminals.
C73X	CDC 73x terminals.
I2740	IBM 2740 terminals.
I3780	IBM 3780 terminals.
I3270	IBM 3270 terminals.

TERMINAL_MODEL (TM)

Specifies a logical name for the type of the terminal in use. This attribute determines what is used for full screen applications such as EDIT_FILE. You may enter a name specifying a terminal definition that you defined, or one of the following system-supplied names:

NOS/VE Model Name	Terminal Model
CDC_721	CDC 721
CDC_722	CDC 722
CDC_722_30	CDC 722_30
CDC_910	CDC 910 workstation
DEC_VT100	DEC VT100 (18 function key definition)
DEC_VT100_GOLD	DEC VT100 (32 function key definition)
DEC_VT220	DEC VT220
IBM_3270	IBM 3270 model 1
IBM_3270_2	IBM 3270 model 2

NOS/VE Model Name	Terminal Model
IBM_3270_3	IBM 3270 model 3
IBM_3270_4	IBM 3270 model 4
IBM_3270_5	IBM 3270 model 5
MAC_CONNECT_10	Apple Macintosh (CONNECT 1.0)
MAC_CONNECT_11	Apple Macintosh (CONNECT 1.1)
MAC_CONNECT_20	Apple Macintosh (CONNECT 2.0)
PC_CONNECT_10	IBM PC (CONNECT 1.0)
PC_CONNECT_11	IBM PC (CONNECT 1.1)
PC_CONNECT_12	IBM PC (CONNECT 1.2)
PC_CONNECT_13	IBM PC (CONNECT 1.3)
SUN_160	Sun Microsystems 160 workstation
TEK_4109	Tektronix 4107 or 4109
TEK_4115	Tektronix 4115 or 4115B
TEK_4125	Tektronix 4125, 4128, or 4129
TV_950	Televideo 950
TV_950_PROTECTED	Televideo 950 with field protection
TV_955	Televideo 955
TV_955_PROTECTED	Televideo 955 with field protection
ZEN_Z19	Zenith Z19 or Heathkit H19
ZEN_Z29	Zenith Z29

NOTE

Changing the terminal model attribute causes a new terminal definition to be loaded the next time a full screen application is initiated. Any `CHANGE_TERMINAL_ATTRIBUTES` command embedded in the terminal definition is executed at this time.

TERMINAL_NAME (TN)

Specifies a unique 31-character name for the terminal in use. This attribute is read-only and you cannot change it. The name is obtained from the network during LOGIN or, on NAMVE/CDCNET connections only, when a detached job is reattached to a terminal. The characters of the default network-assigned name describe the unique physical path through which the terminal is connected to the network. For example, NAM/CCP default terminal names identify the coupler node, MCI node and connection number. The network administrator is responsible for the uniqueness of non-default terminal names.

This attribute is not applicable on INTERCOM as it does not assign terminal names.

TERMINATE_BREAK_CHARACTER (TBC)

Specifies the input character that causes a terminate break condition when it is received as the only character on a line.

Terminal Attribute Defaults

The network establishes the initial value for each terminal attribute.

Terminal Attribute Applicability

The effect of each terminal attribute depends on the network to which the terminal is connected and the type of editing mode in effect.

Network Type

NOS/VE supports terminal access through four networks: NAMVE/CDCNET, NAM/CCP, NAM/CDCNET, and INTERCOM. Not all attributes apply to all networks. Table 13-1 indicates which terminal attributes are supported by each type of network. You can assign a value to any terminal attribute even if it does not apply to a particular network. However, the changed value of a nonapplicable attribute cannot be retrieved and has no effect. The mapping from NOS/VE terminal attributes to NAM/CCP device characteristics is described in the CYBIL File Management manual. This manual also discusses the relationship of NOS/VE attributes to INTERCOM terminal support

Table 13-1. Terminal Attribute Network Applicability

Attribute	NAMVE/ CDCNET	NAM/ CDCNET	NAM/ CCP	INTERCOM
ATTENTION_ CHARACTER	X	na ¹	²	na
BACKSPACE_ CHARACTER	X	X	X	na
BEGIN__LINE_ CHARACTER	X	na	na	na
CANCEL__LINE_ CHARACTER	X	X	X	na
CARRIAGE_ RETURN__DELAY	X	X	X	na
CARRIAGE_ RETURN_ SEQUENCE	X	na	na	na
CHARACTER_ FLOW__CONTROL	X	X	X	na
CODE__SET	X	na	na	na
ECHOPLEX	X	X	X	na
END__LINE_ CHARACTER	X	X	X	na

1. na means not applicable.

2. In this network, the attribute's effect does not correspond to the description given in this chapter. Rather, the value of this attribute qualifies the meaning of the INPUT_EDITING_MODE connection attribute.

(Continued)

Table 13-1. Terminal Attribute Network Applicability (Continued)

Attribute	NAMVE/ CDCNET	NAM/ CDCNET	NAM/ CCP	INTERCOM
END_LINE_POSITIONING	X	X	X	na ¹
END_OUTPUT_SEQUENCE	X	na	na	na
END_PAGE_ACTION	X	na	na	na
END_PARTIAL_CHARACTER	X	X	X	na
END_PARTIAL_POSITIONING	X	X	X	na
FOLD_LINE	X	X	X	na
FORM_FEED_DELAY	X	na	na	na
FORM_FEED_SEQUENCE	X	na	na	na
HOLD_PAGE	X	X	X	na
HOLD_PAGE_OVER	X	na	na	na
LINE_FEED_DELAY	X	X	X	na

1. na means not applicable.

(Continued)

Table 13-1. Terminal Attribute Network Applicability (Continued)

Attribute	NAMVE/ CDCNET	NAM/ CDCNET	NAM/ CCP	INTERCOM
LINE_FEED_SEQUENCE	X	na ¹	na	na
NETWORK_COMMAND_CHARACTER	X	X	X	na
PAGE_LENGTH	X	X	X	X
PAGE_WIDTH	X	X	X	X
PARITY	X	X	X	X
PAUSE_BREAK_CHARACTER	na	na	X	na
STATUS_ACTION	X	X	X	na
TERMINAL_CLASS	na	X	X	X
TERMINAL_MODEL	X	X	X	X
TERMINAL_NAME ²	X	X	X	na
TERMINATE_BREAK_CHARACTER	na	na	X	na

1. na means not applicable.

2. This attribute can only be displayed; it cannot be changed.

Editing Mode

The applicability of certain terminal attributes depends on the editing mode that is in effect (refer to the `INPUT_EDITING_MODE` connection attribute description). Table 13-2 indicates which terminal attributes are in effect for each editing mode. You can change or retrieve any terminal attribute, even if the attribute does not apply to the current editing mode. However, the attribute value has no effect until the editing mode is changed.

See the Terminal Input and Terminal Output sections later in this chapter for a more detailed discussion of editing modes.

Table 13-2. Terminal Attribute Editing Modes

Attribute	Editing Mode
<code>ATTENTION_CHARACTER</code>	Normal/Transparent
<code>BACKSPACE_CHARACTER</code>	Normal
<code>BEGIN_LINE_CHARACTER</code>	Normal
<code>CANCEL_LINE_CHARACTER</code>	Normal
<code>CARRIAGE_RETURN_DELAY</code>	Normal
<code>CARRIAGE_RETURN_SEQUENCE</code>	Normal
<code>CHARACTER_FLOW_CONTROL</code>	Normal/Transparent
<code>CODE_SET</code>	Normal
<code>ECHOPLEX</code>	Normal/Transparent
<code>END_LINE_CHARACTER</code>	Normal
<code>END_LINE_POSITIONING</code>	Normal
<code>END_OUTPUT_SEQUENCE</code>	Normal/Transparent

(Continued)

Table 13-2. Terminal Attribute Editing Modes *(Continued)*

Attribute	Editing Mode
END_PAGE_ACTION	Normal
END_PARTIAL_CHARACTER	Normal
END_PARTIAL_POSITIONING	Normal
FOLD_LINE	Normal
FORM_FEED_DELAY	Normal
FORM_FEED_SEQUENCE	Normal
HOLD_PAGE	Normal
HOLD_PAGE_OVER	Normal
LINE_FEED_DELAY	Normal
LINE_FEED_SEQUENCE	Normal
NETWORK_COMMAND_CHARACTER	Normal
PAGE_LENGTH	Normal
PAGE_WIDTH	Normal
PARITY	Normal/Transparent
PAUSE_BREAK_CHARACTER	Normal/Transparent
STATUS_ACTION	Normal/Transparent
TERMINAL_CLASS	Normal/Transparent
TERMINAL_MODEL	Normal/Transparent
TERMINAL_NAME	Normal/Transparent
TERMINATE_BREAK_CHARACTER	Normal/Transparent

Managing Connection Attributes

The set of connection attribute values that actually controls the terminal interaction for a connection is maintained by the network. However, NOS/VE maintains a separate set of connection attribute values for each instance of open of a file associated with a terminal connection. See the CYBIL File Management manual for more information about attribute values at the instance-of-open level.

Connection Attribute Levels

To simplify the specification of connection attribute values to be used for an instance of open, NOS/VE maintains three levels of connection attribute values. Each level specifies the default or initial values to be used when an instance of the next level is created.

Level	Description
Default	Specifies the default attribute values assigned to a file when it is associated with a terminal. (For the initial values at this level, see Initial Default Values for Connection Attributes later in this chapter.) If an attribute value is not specified when a terminal file is created (via the REQUEST_TERMINAL command), the initial attribute value at the file level is the corresponding value from the default level. Changes at the default level do not affect values at the file level for existing files.
File	Specifies the initial attribute values to be used for an instance of open of a terminal file. Changes at the file level do not affect values at the instance-of-open level for existing instances of open. (SCL commands do not affect attribute values at the instance-of-open level. See the CYBIL File Management manual.)
Instance of Open	Specifies the attribute values to be used on a terminal connection when the connection is accessed via a particular instance of open. NOS/VE monitors the attribute values currently in effect for a connection. Each time a task accesses a connection, attribute values that do not match the specified instance-of-open attributes are changed.

SCL Commands

The following SCL commands allow you to change or retrieve connection attributes at the default and file levels:

Level	SCL Command
Default	CHANGE_TERM_CONN_DEFAULT DISPLAY_TERM_CONN_DEFAULT
File	CHANGE_CONNECTION_ATTRIBUTE DISPLAY_CONNECTION_ATTRIBUTE

The CHANGE_TERM_CONN_DEFAULT command enables you to change the connection attribute defaults for terminal connection. This command does not affect connection attribute values at the file level for existing files.

Refer to the Connection Attribute Descriptions section for attributes, names, and descriptions.

The DISPLAY_TERM_CONN_DEFAULT command allows you to display the connection attribute defaults for a terminal connection.

The CHANGE_CONNECTION_ATTRIBUTE command lets you change the terminal file's connection attributes. The command only changes the connection attribute values at the file level for the specified file. It does not affect connection attribute values at the instance of open level for existing instances of open of the specified file.

The DISPLAY_CONNECTION_ATTRIBUTE command enables you to display either all or a specified set of the terminal file's connection attributes.

Connection Attribute Set

Connection attributes describe the characteristics of a particular network's interactive environment. Once a terminal is connected to a network, the values specified for a connection attribute will affect the connection. The following brief descriptions of each attribute include:

- The attribute's parameter name and abbreviation.
- The attribute's purpose.
- Valid attribute values.

For more detailed information, see the CDCNET Terminal Interface Usage manual.

ATTENTION_CHARACTER_ACTION (ACA)

Specifies the type of user interrupt command simulated by the network when the character defined by the ATTENTION_CHARACTER terminal attribute is received from the terminal. If the ATTENTION_CHARACTER is set to NUL, this attribute has no effect. The following values are allowed:

0

All typed-ahead input is discarded.

1

All undelivered input and output is discarded, and a pause break condition is raised.

2 - 9

All undelivered input and output is discarded, and a terminate break condition is raised.

BREAK_KEY_ACTION (BKA)

Specifies the type of user interrupt command simulated by the network when the break key is pressed at the terminal. The following values are allowed:

0

All typed-ahead input is discarded.

1

All undelivered input and output is discarded, and a pause break condition is raised.

2 - 9

All undelivered input and output is discarded, and a terminate break condition is raised.

END_OF_INFORMATION (EOI)

Specifies a string of 0 to 31 characters that, when entered as a complete input line, is interpreted as an end-of-information mark on the input file. A string of zero characters indicates that EOI is never reached for the terminal file. This attribute is for NOS/VE internal use only; it is not sent to the network.

INPUT_BLOCK_SIZE (IBS)

Specifies the maximum number of characters (80 through 2000) stored by the network before input data is forwarded to NOS/VE. When the input data is forwarded, it is sent as a partial input line.

INPUT_EDITING_MODE (IEM)

Specifies how the network edits the data received from the terminal. (Refer to the Terminal Input section of this chapter for more information concerning the input editing modes.) The following values are allowed:

NORMAL

NORMAL editing mode is in effect. The network edits input to remove special codes or characters before the input is forwarded to NOS/VE.

TRANSPARENT

TRANSPARENT editing mode is in effect. The network forwards input to NOS/VE without converting or deleting any special codes or characters.

NOTE

The value of the INPUT_EDITING_MODE attribute also determines how the network edits output sent to the terminal.

INPUT_OUTPUT_MODE (IOM)

Specifies how the network coordinates the terminal input and output streams. The following values are allowed:

UNSOLICITED (U)

The network edits and forwards input data as it is received. Input need not be solicited by a task in order to be edited and forwarded.

If output is received while an input line is being entered, the output is not sent to the terminal until the input line is completed. If an input line is started while output is being sent to the terminal, the output is suspended until the input line is completed.

Note that typed-ahead input may be edited in the wrong mode if the INPUT_EDITING_MODE is changed.

SOLICITED (S)

The network does not edit or forward input data until it is solicited by a task. A task solicits input by reading from a terminal file associated with the terminal.

When input has been solicited, input and output are coordinated the same way as for UNSOLICITED_OUTPUT mode. If output is received while an input line is being entered, the output is not sent to the terminal until the input line is completed. If an input line is started while output is being sent to the terminal, the output is suspended until the input line is completed.

When input has not been solicited, input is accepted by the network and stored in raw form. The input data is not edited or forwarded until input is solicited. Echoplexing, backspacing, and cursor positioning are not performed. Output is sent to the terminal as it is received, and received input does not suspend output in progress.

Note that typed-ahead input is always edited in the desired mode if the INPUT_EDITING_MODE is changed.

FULL_DUPLEX (F)

The network does not coordinate the input and output streams. Input data is edited and forwarded as it is received. Output data is sent to the terminal as it is received.

Note that typed-ahead input may be edited in the wrong mode if the INPUT_EDITING_MODE is changed.

INPUT_TIMEOUT (IT)

Specifies whether NOS/VE is to limit the amount of time a task waits for input from the terminal when it reads from a terminal file. The length of the timeout interval and the action taken when a timeout occurs are determined by the INPUT_TIMEOUT_LENGTH and INPUT_TIMEOUT_PURGE connection attributes.

TRUE

NOS/VE limits the task wait time.

FALSE

NOS/VE does not limit the task wait time

This attribute is for NOS/VE internal use only; it is not sent to the network.

INPUT_TIMEOUT_LENGTH (ITL)

Specifies the maximum number of milliseconds (0 through 86,401) that a task is to wait for input from a terminal when it reads from a terminal file. If no input occurs within the specified timeout interval, a GET call returns abnormal status. This attribute's effect is conditioned by the value of the INPUT_TIMEOUT connection attribute. This attribute is for NOS/VE internal use only; it is not sent to the network.

INPUT_TIMEOUT_PURGE (ITP)

Specifies whether undelivered terminal input and output is to be discarded when an input timeout condition occurs. This attribute has no effect if the INPUT_TIMEOUT_LENGTH attribute is set to 0. This attribute's effect is conditioned by the value of the INPUT_TIMEOUT connection attribute. This attribute is for NOS/VE internal use only; it is not sent to the network.

PARTIAL_CHARACTER_FORWARDING (PCF)

Specifies whether the network forwards a partial input line when an END_PARTIAL_CHARACTER is received from the terminal. (The END_PARTIAL_CHARACTER terminal attribute is described earlier in this chapter.)

TRUE

The network sends a partial input line to NOS/VE upon receipt of the END_PARTIAL_CHARACTER.

FALSE

The network stores the END_PARTIAL_CHARACTER as part of the data to be forwarded to NOS/VE. A partial input line is not sent.

PROMPT_FILE (PF)

Specifies the local file name of the file to which the prompt string is written. NOS/VE opens the file specified by the PROMPT_FILE attribute when a task performs its first input from the terminal file. This attribute is for NOS/VE internal use only; it is not sent to the network.

PROMPT_STRING (PS)

Specifies the string written to the prompt file when a task reads from the terminal file. This attribute is for NOS/VE internal use only; it is not sent to the network.

STORE_BACKSPACE_CHARACTER (SBC)

Specifies how the network processes the character specified as the BACKSPACE_CHARACTER when it is received from the terminal. (The BACKSPACE_CHARACTER terminal attribute is described earlier in this chapter.)

TRUE

The network stores the BACKSPACE_CHARACTER as part of the data to be forwarded to NOS/VE.

FALSE

The network discards the BACKSPACE_CHARACTER and removes the last character from the data to be forwarded to NOS/VE.

STORE_NUL_DELS (SND)

Specifies whether the network stores or discards the NUL and DEL characters when they are received from the terminal.

TRUE

The network stores the NUL or DEL characters as part of the data to be forwarded to NOS/VE.

FALSE

The network discards the NUL and DEL characters.

TRANSPARENT_CHARACTER_MODE (TCM)

Specifies the action the network takes when a **TRANSPARENT_FORWARD_CHARACTER** or a **TRANSPARENT_TERMINATE_CHARACTER** is received from the terminal. The following values are allowed:

FORWARD (F)

The network forwards the stored input characters as a complete input line to NOS/VE when a **TRANSPARENT_FORWARD_CHARACTER** is received from the terminal. The **TRANSPARENT_FORWARD_CHARACTER** is not included in the input line. **TRANSPARENT** editing mode remains in effect.

TERMINATE (T)

The network forwards the stored input characters as a complete input line to NOS/VE and reverts to **NORMAL** editing mode when a **TRANSPARENT_TERMINATE_CHARACTER** is received from the terminal. The **TRANSPARENT_TERMINATE_CHARACTER** is not included in the input line.

FORWARD_TERMINATE (FT)

The network forwards the stored input characters as a complete input line to NOS/VE when a **TRANSPARENT_FORWARD_CHARACTER** is received from the terminal. The **TRANSPARENT_FORWARD_CHARACTER** is not included in the input line. The network reverts to **NORMAL** editing mode when a **TRANSPARENT_TERMINATE_CHARACTER** is received after a **TRANSPARENT_FORWARD_CHARACTER**.

NONE (N)

The network takes no action when a **TRANSPARENT_FORWARD_CHARACTER** or **TRANSPARENT_TERMINATE_CHARACTER** is received from the terminal.

TRANSPARENT_FORWARD_CHARACTER (TFC)

Specifies a string of up to four characters; any one of these characters is recognized by the network as the transparent mode forwarding character. See the description of the TRANSPARENT_CHARACTER_MODE connection attribute for more information.

TRANSPARENT_LENGTH_MODE (TLM)

Specifies the action the network takes when it receives the number of characters specified by the TRANSPARENT_MESSAGE_LENGTH connection attribute. The following are allowed values:

FORWARD (F)

The network forwards the stored input characters as a complete input line to NOS/VE when the number of characters specified by the TRANSPARENT_MESSAGE_LENGTH attribute has been received. The input line may exceed the specified length if more input data is available by the time the line is actually forwarded.

FORWARD_EXACT (FE)

The network forwards the stored input characters as a complete input line to NOS/VE when the number of characters specified by the TRANSPARENT_MESSAGE_LENGTH attribute has been received. The length of the input line will equal the specified length.

TERMINATE (T)

The network forwards the stored input characters as a complete input line to NOS/VE and reverts to NORMAL editing mode when the number of characters specified by the TRANSPARENT_MESSAGE_LENGTH attribute has been received.

NONE (N)

The network takes no action when the number of characters specified by the TRANSPARENT_MESSAGE_LENGTH attribute has been received.

TRANSPARENT_MESSAGE_LENGTH (TML)

Specifies the minimum number of characters (1 through 32,767) forwarded in each transparent input message. See the description of the **TRANSPARENT_LENGTH_MODE** connection attribute for more information.

TRANSPARENT_TERMINATE_CHARACTER (TTC)

Specifies a string of up to four characters; any one of these characters is recognized by the network as the transparent mode terminating character. See the description of the **TRANSPARENT_CHARACTER_MODE** connection attribute for more information.

TRANSPARENT_TIMEOUT_MODE (TTM)

Specifies the action the network takes when no input is received from the terminal for an interval of 400 milliseconds or more. The following are allowed values:

FORWARD (F)

The network forwards the stored input characters as a complete input line to NOS/VE when a timeout occurs between characters.

TERMINATE (T)

The network reverts to **NORMAL** editing mode when a timeout occurs between characters.

NONE (N)

No action is taken when a timeout occurs between characters.

Initial Default Values for Connection Attributes

The network establishes the initial value for each connection attribute. However, the initial connection attribute values maintained by the network are irrelevant since NOS/VE changes the network-maintained values to the corresponding instance-of-open values on the first access to the connection.

NOS/VE establishes the set of initial values for connection attributes at the default level. These initial values are specified in table 13-3. If connection attribute values are not explicitly specified at either the default or file level, these initial values become the connection attributes for a terminal file.

Table 13-3. NOS/VE Initial Values at the Default Level

Connection Attribute	Initial Value
ATTENTION_CHARACTER_ACTION	2
BREAK_KEY_ACTION	0
END_OF_INFORMATION	'*EOI'
INPUT_BLOCK_SIZE	160
INPUT_EDITING_MODE	NORMAL
INPUT_OUTPUT_MODE	UNSOLICITED
INPUT_TIMEOUT	FALSE
INPUT_TIMEOUT_LENGTH	86401
INPUT_TIMEOUT_PURGE	TRUE
PARTIAL_CHARACTER_FORWARDING	FALSE

(Continued)

Table 13-3. NOS/VE Initial Values at the Default Level
(Continued)

Connection Attribute	Initial Value
PROMPT_FILE	\$local.output
PROMPT_STRING	' ? '
STORE_BACKSPACE_CHARACTER	FALSE
STORE_NULS_DELS	FALSE
TRANSPARENT_CHARACTER_MODE	NONE
TRANSPARENT_FORWARD_CHARACTER	Carriage Return
TRANSPARENT_LENGTH_MODE	NONE
TRANSPARENT_MESSAGE_LENGTH	2000
TRANSPARENT_TERMINATE_CHARACTER	Carriage Return
TRANSPARENT_TIMEOUT_MODE	NONE

Connection Attribute Applicability

The effect of each connection attribute depends on the network used to access the terminal and the type of editing mode that is in effect.

Network Type

NOS/VE supports terminal access through the following four networks: NAMVE/CDCNET, NAM/CCP, NAM/CDCNET, and INTERCOM. Not all attributes apply to all networks. Table 13-4 indicates which connection attributes are applicable for each network. You can assign a value to any connection attribute even if does not apply to a particular network. However, the changed value of a nonapplicable attribute cannot be retrieved and has no effect.

For detailed information on the mapping from NOS/VE terminal and connection attributes to NAM/CCP device characteristics, and on the relationship between NOS/VE connection attributes and INTERCOM terminal support, see the CYBIL File Management manual, listed in appendix B of this manual.

Table 13-4. Connection Attribute Network Applicability

Attribute	NAMVE/ CDCNET	NAM/ CDCNET	NAM/ CCP	INTERCOM
ATTENTION_ CHARACTER_ ACTION	X	na ¹	na	na
BREAK_KEY_ ACTION	X	X	X	na
END_OF_ INFORMATION	X	X	X	X
INPUT_BLOCK_ SIZE	X	na	na	na
INPUT_EDITING_ MODE	X	X	X	X

1. na means not applicable.

(Continued)

Table 13-4. Connection Attribute Network Applicability
(Continued)

Attribute	NAMVE/ CDCNET	NAM/ CDCNET	NAM/ CCP	INTERCOM
INPUT_OUTPUT_ MODE	X	X	X	na
INPUT_TIMEOUT	X	X	X	X
INPUT_TIMEOUT_ LENGTH	X	X	X	X
INPUT_TIMEOUT_ PURGE	X	X	X	X
PARTIAL_ CHARACTER_ FORWARDING	X	X	X	na
PROMPT_FILE	X	X	X	X
PROMPT_FILE_ID	X	X	X	X
PROMPT_STRING	X	X	X	X
STORE_ BACKSPACE_ CHARACTER	X	X	X	na
STORE_NULS_ DELS	X	X	X	na

1. na means not applicable.

(Continued)

Table 13-4. Connection Attribute Network Applicability
(Continued)

Attribute	NAMVE/ CDCNET	NAM/ CDCNET	NAM/ CCP	INTERCOM
TRANSPARENT_ CHARACTER_ MODE	X	X	X	na
TRANSPARENT_ FORWARD_ CHARACTER	X	X	X	na
TRANSPARENT_ LENGTH_ MODE	X	X	X	na
TRANSPARENT_ MESSAGE_ LENGTH	X	X	X	na
TRANSPARENT_ TERMINATE_ CHARACTER	X	X	X	na
TRANSPARENT_ TIMEOUT_ MODE	X	X	X	na

1. na means not applicable.

Editing Mode

The applicability of many connection attributes depends on the editing mode in effect (refer to the INPUT_EDITING_MODE connection attribute description). Table 13-5 indicates which connection attributes are in effect for each editing mode. You can change or retrieve any connection attribute even if the attribute does not apply to the current editing mode. However, the attribute value has no effect until the editing mode is changed.

See the Terminal Input and Terminal Output sections later in this chapter for a more detailed discussion of editing modes.

Table 13-5. Connection Attribute Editing Modes

Attribute	Editing Mode
ATTENTION_CHARACTER_ACTION	Normal/Transparent
BREAK_KEY_ACTION	Normal/Transparent
END_OF_INFORMATION	Normal
INPUT_BLOCK_SIZE	Normal/Transparent
INPUT_EDITING_MODE	Normal/Transparent
INPUT_OUTPUT_MODE	Normal/Transparent
INPUT_TIMEOUT	Normal/Transparent
INPUT_TIMEOUT_LENGTH	Normal/Transparent
INPUT_TIMEOUT_PURGE	Normal/Transparent
PARTIAL_CHARACTER_FORWARDING	Normal
PROMPT_FILE	Normal
PROMPT_FILE_ID	Normal

(Continued)

Table 13-5. Connection Attribute Editing Modes (Continued)

Attribute	Editing Mode
PROMPT_STRING	Normal
STORE_BACKSPACE_CHARACTER	Normal
STORE_NULS_DELS	Normal
TRANSPARENT_CHARACTER_MODE	Transparent
TRANSPARENT_FORWARD_CHARACTER	Transparent
TRANSPARENT_LENGTH_MODE	Transparent
TRANSPARENT_MESSAGE_LENGTH	Transparent
TRANSPARENT_TERMINATE_CHARACTER	Transparent
TRANSPARENT_TIMEOUT_MODE	Transparent

Terminal Input

NOS/VE allows more than one task in a job to be reading from the same terminal at the same time. When a task reads from a terminal file, a complete input line is delivered to the task. When multiple tasks are waiting for input at the same time, the order in which input lines are delivered to tasks does not necessarily match the order in which the tasks issued calls to read from the terminal.

Input Buffering

Input characters received from a terminal are buffered by the network and forwarded to NOS/VE under conditions defined by certain terminal and connection attributes. The attributes that define forwarding conditions vary depending on the editing mode in effect. The forwarded data is buffered by NOS/VE until a task reads from a file associated with the terminal.

Input data is forwarded to NOS/VE as either a complete input line or a partial input line. Partial input lines are concatenated by NOS/VE to form a complete line before input data is delivered to a task. In the following example, two lines are entered in normal editing mode; the first line ends with the `END_PARTIAL_CHARACTER`, and the second line ends with the `END_LINE_CHARACTER`:

```
<line 1>end_partial_character  
<line 2>end_line_character
```

The input is then delivered to a task which reads from the terminal as a single line as follows:

```
<line 1><line 2>
```

Normal Editing Mode

If the `INPUT_EDITING_MODE` connection attribute is normal, input lines consist of characters from the ASCII 128-character set. The network performs the following editing actions on input characters as they are received from the terminal:

Editing Action	Description
Parity checking	The network verifies that the parity bit in each character is set as determined by the <code>PARITY</code> terminal attribute. After verifying the parity bit, the network sets the bit to 0 (even if the <code>PARITY</code> attribute is set to no parity).
Echoplexing	The network optionally sends (echoes) received characters back to the terminal. This action is conditioned by the <code>ECHOPLEX</code> terminal attribute.
Character code conversion	The network converts each character from the terminal's encoding to the appropriate ASCII 128-character code, as determined by the <code>CODE_SET</code> terminal attribute.
Store and forward	The network stores characters received from the terminal until a forwarding condition occurs. Forwarding conditions are defined by the <code>END_LINE_CHARACTER</code> and <code>END_PARTIAL_CHARACTER</code> terminal attributes and the <code>PARTIAL_CHARACTER_FORWARDING</code> and <code>INPUT_BLOCK_SIZE</code> connection attributes.
Backspacing	The network optionally discards the last character in the store and forward buffer when the <code>BACKSPACE_CHARACTER</code> is received from the terminal. This action is conditioned by the <code>STORE_BACKSPACE_CHARACTER</code> connection attribute.

Editing Action	Description
Cursor positioning	The network positions the cursor on the terminal display when certain characters are received. Cursor positioning actions are defined by the <code>END_LINE_POSITIONING</code> and <code>END_PARTIAL_POSITIONING</code> terminal attributes.
Network command recognition	The network recognizes an input line which begins with the <code>NETWORK_COMMAND_CHARACTER</code> as a network command, and processes the command rather than forwarding the input line.
Attention character action	The network takes the action specified by the <code>ATTENTION_CHARACTER_ACTION</code> connection attribute whenever the <code>ATTENTION_CHARACTER</code> is received from the terminal.
Break key action	The network takes the action specified by the <code>BREAK_KEY_ACTION</code> connection attribute whenever the break key is pressed.
Character discarding	The network optionally discards certain characters when they are received from the terminal. Character discarding actions are conditioned by the <code>STORE_NULS_DELS</code> connection attribute, and the <code>BEGIN_LINE_CHARACTER</code> and <code>CANCEL_LINE_CHARACTER</code> terminal attributes.

Transparent Editing Mode

If the `INPUT_EDITING_MODE` is transparent, input lines consist of characters as they are encoded by the terminal. The network performs the following minimal editing actions on input characters as they are received from the terminal:

Editing Action	Description
Parity checking	The network verifies that the parity bit in each character is set as determined by the <code>PARITY</code> terminal attribute. If the <code>PARITY</code> terminal attribute is set to no parity, the network does not change the parity bit in input characters, which allows input data to include 256 distinct characters. Otherwise, the network sets the parity bit to 0 in each character, which restricts input data to 128 distinct characters.
Echoplexing	The network optionally sends (echoes) received characters back to the terminal. This action is conditioned by the <code>ECHOPLEX</code> terminal attribute.
Store and forward	The network stores characters received from the terminal until a forwarding condition occurs. Forwarding conditions are defined by the following connection attributes: <code>TRANSPARENT_CHARACTER_MODE</code> , <code>TRANSPARENT_LENGTH_MODE</code> , <code>TRANSPARENT_TIMEOUT_MODE</code> , and <code>INPUT_BLOCK_SIZE</code> .

Editing Action	Description
Attention character action	The network takes the action specified by the <code>ATTENTION_CHARACTER_ACTION</code> connection attribute whenever the <code>ATTENTION_CHARACTER</code> is received from the terminal.
Break key action	The network takes the action specified by the <code>BREAK_KEY_ACTION</code> connection attribute whenever the break key is pressed at the terminal.

Typed-Ahead Input

Input data may be entered (typed ahead) before a task reads from the terminal. The network and NOS/VE buffer the typed-ahead input until a task requests the input. The amount of input that can be typed ahead depends on the buffering capacity of the network:

- When you reach the type-ahead limit on a NAMVE/CDCNET or NAM/CDCNET connection, the network sends the BEL character in response to received characters and discards the characters. User interrupts are suspended; however, the BREAK key and ATTENTION_CHARACTER key are still enabled.
- When you reach the type-ahead limit on a NAM/CCP connection, the network sends the following message:

WAIT..

You must then wait until a task reads the input which has been entered. You cannot interrupt the task with a PAUSE_BREAK or TERMINATE_BREAK character until the task accepts the input.

Terminal Output

NOS/VE allows more than one task in a job to be writing to the same terminal at the same time.

NOS/VE buffers output data so that it can be sent to the terminal in large amounts. The following concepts are used to describe terminal output processing:

- An *output record* is a sequence of characters which are to be sent to the terminal as a unit. In the case of normal output editing, an output record corresponds to a line to be displayed at the terminal.
- An *output message* consists of one or more output records.
- An *output block* consists of one or more output records sent to the terminal as one network message. An output block may be a complete output message or a partial output message. An output record may not span output blocks.

Normal Editing Mode

If the INPUT_EDITING_MODE connection attribute is normal, each output record corresponds to a line of output. An output line consists of 0 to 2,043 characters in the ASCII 128-character set.

Output lines within an output block are separated by an ASCII US character, which is inserted by NOS/VE when a line is added to a block. This character is not sent to the terminal by the network. Therefore, the ASCII US character should not be included in data written to the terminal. If the ASCII US character is included in terminal output data, it will be interpreted as an end-of-line by the network.

The network performs the following processing on output lines as they are sent to the terminal:

Output Processing	Description
Interline positioning	The network determines the cursor positioning which occurs between output lines by either processing format effectors imbedded in the output data or following a fixed positioning convention.
Parity generation	If the PARITY terminal attribute is set to no parity, the network sets the parity bit in each character to 0. Otherwise, the network sets the parity bit in each character as determined by the PARITY terminal attribute.
Character code conversion	The network converts each ASCII 128-character code to the appropriate terminal code, as determined by the CODE_SET terminal attribute.
Page holding	The network optionally holds each page of uninterrupted (by input) output on the terminal display for examination by the terminal user before displaying further output. This action is controlled by the HOLD_PAGE, HOLD_PAGE_OVER, and PAGE_LENGTH terminal attributes.
Line folding	The network optionally splits an output line which exceeds the terminal's page width into multiple physical lines. This action is controlled by the FOLD_LINE terminal attribute.

Output Processing	Description
Page boundary form feed	The network optionally performs a form feed action at the end of every page of uninterrupted (by input) output. This action is controlled by the <code>END_PAGE_ACTION</code> and <code>PAGE_LENGTH</code> terminal attributes.
Output message boundary notification	The network sends the characters defined by the <code>END_OUTPUT_SEQUENCE</code> terminal attribute to the terminal at the end of every output message.

Interline Positioning

When the `FILE_CONTENTS` file attribute of a terminal file is set to `LIST`, the network processes the first character of each output line as a format effector. The value of the format effector determines what interline cursor positioning is to occur. The format effector character is not sent to the terminal.

When the `FILE_CONTENTS` file attribute of a terminal file is not set to `LIST`, the network treats the first character of each output line as the first character to be displayed. The network positions the cursor at the beginning of the next line.

Page Holding

Page holding is enabled whenever the `HOLD_PAGE` terminal attribute has a value of `TRUE`, and the `PAGE_LENGTH` terminal attribute has a nonzero value. When page holding is enabled, the network suspends output after sending a page of output which has not been interrupted by input. This allows the terminal user to examine the displayed information before further output is displayed.

In order to resume the display of output following suspension, the terminal user must enter a line of input. If a nonempty line is entered, it is sent to NOS/VE as input data. If an empty line is entered, it is not sent to NOS/VE.

If the HOLD_PAGE_OVER terminal attribute is set to TRUE, the network sends a message whenever output is suspended.

- CDCNET sends the following message:

<OVER>

- NAM/CCP sends the following message:

OVER..

The message is a visual prompt that indicates that the network is waiting for input before "turning" to the next page.

Transparent Editing Mode

If the INPUT_EDITING_MODE connection attribute is transparent, no significance is assigned to the grouping of output data into output records. Each output record is delivered to the terminal as a stream of raw characters. No action is taken as a result of output record boundaries, and no constraint exists on the size of output records.

The network processes output records as they are sent to the terminal for display in the following manner:

- If the parity terminal attribute is set to no parity, the network does not change the parity bit in the character sent to the terminal, which allows output data to include 256 characters.
- If the parity terminal attribute is set to a value other than no parity, the network sets the parity bit as determined by the parity terminal attribute in each character sent to the terminal, which restricts output data to 128 characters.

Page Width and Page Length Attributes

There are both terminal and file attributes for page width and page length. The two types of attributes are used for different purposes and may have different values.

The terminal attributes define the physical characteristics of the terminal, and control the line folding and page holding actions.

The file attributes define the logical characteristics of the information written to the terminal. Tasks which produce output (for terminals as well as any other device), should query the file attributes and tailor the generated output to the reported dimensions. If a value has not been specified for either of the file attributes, the corresponding terminal attribute value is used as a default.

Using CONNECT	14-1
Saving NOS/VE File Attributes During File Transfers	14-1
Example: Using CONNECT to Transfer an Object Library	14-4
Using XMODEM	14-6
Parity Requirements	14-6
File Types	14-6
Micro Binary Files	14-6
Text Files	14-7
CYBER Binary Files	14-7
Special Characters	14-8
Setting the High-Order Bit	14-9
Initiating the File Transfer	14-9
Specifying File Name	14-10
Establishing Transfer Direction	14-10
Specifying File Type	14-11
Specifying the End-of-Line	14-13
Specifying the File Type Block	14-13
Checking for Errors	14-14
Specifying File Markers	14-14
Specifying a Configuration File	14-16
KERMIT-VE	14-18
Accessing the KERMIT online manual	14-18
Simple KERMIT File Transfers	14-19
Desktop/VE	14-21

NOS/VE supports several methods for transferring files between a microcomputer and a host running NOS/VE:

- CDC CONNECT
- XMODEM command
- KERMIT-VE
- Desktop/VE for users of the Apple Macintosh

Using CONNECT

If you have access to CONNECT, refer to the CONNECT manual for your microcomputer to determine if the version of CONNECT you are using supports use with NOS/VE. (Versions of CYBERNET CONNECT purchased from CDC CYBERNET and Control Data CONNECT for the CDC 110 will not work on NOS/VE.)

Saving NOS/VE File Attributes During File Transfers

When you transfer a file to NOS/VE using CONNECT, CONNECT sets the attributes of the file to certain defaults depending upon whether the file was transferred in ASCII or binary mode. Files transferred in ASCII mode have the following attributes:

Access_Mode	:	(read, shorten, append, modify, execute)
Application_Information	:	none
Average_Record_Length	:	1
Block_Type	:	system_specified
Character_Conversion	:	no
Collate_Table_Name	:	none
Compression_Procedure_Name	:	none
Data_Padding	:	0
Dynamic_Home_Block_Space	:	no
Embedded_Key	:	yes
Error_Exit_Procedure_Name	:	none
Error_Limit	:	0
Estimated_Record_Count	:	0
File_Access_Procedure_Name	:	none
File_Contents	:	unknown

Using CONNECT

```
File_Label_Type      : unlabelled
File_Limit           : 4398046511103
File_Organization   : sequential
File_Processor       : unknown
File_Structure       : unknown
Forced_Write         : no
Global_Access_Mode   : (read, shorten, append, modify,
                        execute)
Global_File_Address  : 0
Global_File_Name     : $766461040S0102D19880112T025000
Global_File_Position : boi
Global_Share_Mode    : none
Hashing_Procedure_Name : (amp$system_hashing_procedure)
Index_Levels         : 2
Index_Padding        : 0
Initial_Home_Block_Count : 1
Internal_Code        : ascii
Key_Length           : 1
Key_Position         : 0
Key_Type             : uncollated
Line_Number          : ("Location" 1, "Length" 1)
Loading_Factor       : 90
Lock_Expiration_Time : 60000
Logging_Options      : none
Log_Residence        : none
Maximum_Block_Length : 4128
Maximum_Record_Length : 256
Message_Control      : none
Minimum_Block_Length : 18
Minimum_Record_Length : 0
Open_Position        : $boi
Padding_Character    : ' '
Page_Format          : burstable
Page_Length          : 60
Page_Width           : 132
Permanent            : yes
Preset_Value         : 0
Record_Limit         : 4398046511103
Record_Type          : variable
Records_Per_Block    : 65535
Ring_Attributes      : (11, 11, 11)
Size                 : 44
Statement_Identifier : ("Length" 1, "Location" 1)
User_Information     : none
```

Files transferred in binary mode have the following attributes:

Access_Mode	: (read, shorten, append, modify, execute)
Application_Information	: none
Average_Record_Length	: 1
Block_Type	: system_specified
Character_Conversion	: no
Collate_Table_Name	: none
Compression_Procedure_Name	: none
Data_Padding	: 0
Dynamic_Home_Block_Space	: no
Embedded_Key	: yes
Error_Exit_Procedure_Name	: none
Error_Limit	: 0
Estimated_Record_Count	: 0
File_Access_Procedure_Name	: none
File_Contents	: object
File_Label_Type	: unlabelled
File_Limit	: 4398046511103
File_Organization	: sequential
File_Processor	: unknown
File_Structure	: unknown
Forced_Write	: no
Global_Access_Mode	: (read, shorten, append, modify, execute)
Global_File_Address	: 0
Global_File_Name	: \$247564340S0102D19880112T025025
Global_File_Position	: boi
Global_Share_Mode	: none
Hashing_Procedure_Name	: (amp\$system_hashing_procedure)
Index_Levels	: 2
Index_Padding	: 0
Initial_Home_Block_Count	: 1
Internal_Code	: ascii
Key_Length	: 1
Key_Position	: 0
Key_Type	: uncollated
Line_Number	: ("Location" 1, "Length" 1)
Loading_Factor	: 90
Lock_Expiration_Time	: 60000
Logging_Options	: none
Log_Residence	: none
Maximum_Block_Length	: 4128
Maximum_Record_Length	: 256
Message_Control	: none

```
Minimum_Block_Length      : 18
Minimum_Record_Length     : 0
Open_Position             : $boi
Padding_Character         : ' '
Page_Format               : burstable
Page_Length               : 60
Page_Width                : 132
Permanent                 : yes
Preset_Value              : 0
Record_Limit              : 4398046511103
Record_Type               : undefined
Records_Per_Block         : 65535
Ring_Attributes           : (11, 11, 11)
Size                      : 18
Statement_Identifier       : ("Length" 1, "Location" 1)
User_Information           : none
```

If you have created a file on NOS/VE with attributes other than the defaults listed above, and you are going to transfer that file to and from a microcomputer using CONNECT, you may want to save the file's attributes. To save NOS/VE file attributes when transferring NOS/VE files to and from a microcomputer, use the permanent file backup and restore utilities.

Example: Using CONNECT to Transfer an Object Library

The following example details a procedure for using CONNECT and the permanent file backup and restore utilities to transfer a NOS/VE object library called :NVE1.ME.COMMAND_LIBRARY from a NOS/VE system to a microcomputer and back again.

NOTE

To successfully use the steps in the following example, you must be running either CDC CONNECT for the IBM Personal Computer¹ version 1.15 (or greater) or CDC CONNECT for the Macintosh² version 1.0 (or greater).

1. Registered trademark of the International Business Machines Corporation.

2. Registered trademark of Apple Computer, Inc.

1. Create a local backup copy of file :NVE1.ME.COMMAND_LIBRARY using the backup utility. Enter these commands on NOS/VE:

```
/backup_permanent_file backup_file=$local.backup_copy ..
../list=$output
PUB/backup_file file=:nve1.me.command_library
PUB/quit
```

2. Use CONNECT to perform a binary transfer of the file \$LOCAL.BACKUP_COPY to the micro.
3. When you want to transfer the microcomputer copy of \$LOCAL.BACKUP_COPY back to NOS/VE, first create a NOS/VE file with a RECORD_TYPE of VARIABLE to accept the microcomputer file. For instance, assuming you name this file NEW_FILE, enter the following command on NOS/VE:

```
/set_file_attributes file=$local.new_file ..
../record_type=variable
```

4. Use CONNECT to send the microcomputer copy of \$LOCAL.BACKUP_COPY to the NOS/VE file NEW_FILE that you created in the previous step.
5. To restore the file attributes and name the file \$USER.NEW_COMMAND_LIBRARY, enter the following commands on NOS/VE:

```
/restore_permanent_file list=$output
PUR/restore_file file=:nve1.me.command_library ..
PUR../backup_file=$local.new_file ..
PUR../new_file_name=$user.new_command_library
PUR/quit
```

Using XMODEM

The XMODEM command transfers files between a host running NOS/VE and a microcomputer using the Christensen protocol. (Because NOS/BE INTERCOM does not support packet forwarding as required by XMODEM, you cannot use XMODEM when you are connected to NOS/VE by INTERCOM.)

Parity Requirements

To use the Christensen protocol, you must be able to set PARITY to NONE. The Christensen protocol is an 8-bit protocol that requires all 8 bits to be passed as data. Data networks that cannot be set to a parity of none may not be used. XMODEM automatically sets PARITY to NONE before a transfer, and restores it to its original value before ending.

File Types

When transferring a file from a host to a microcomputer, file attributes determine the type of transfer that is done. When transferring a file from a microcomputer to a host, file attributes are set when the file is transferred. You can transfer the following types of files:

- Micro binary files
- Text files
- CYBER binary files

Micro Binary Files

A micro binary file is indicated when the USER_INFORMATION file attribute is set to MICRO_BINARY. This file attribute may be set by XMODEM or by the SET_FILE_ATTRIBUTE command.

A micro binary file is read and written character for character (characters are 8 bits). A NOS/VE file with the USER_INFORMATION file attribute set to MICRO_BINARY is always transferred from the host to the microcomputer in this manner regardless of any other file attributes.

Text Files

If a NOS/VE file has a `RECORD_TYPE` file attribute of `VARIABLE` and the `USER_INFORMATION` file attribute is not `MICRO_BINARY`, the file is transferred to a microcomputer as a text file. Text files are read and written a line at a time.

When a text file is sent to a microcomputer, an ASCII carriage return or carriage return and line feed are added to the end of each line. Which character or characters are added depends on the value indicated on the `LINE_FEED` parameter of the XMODEM command. During a transfer from host to microcomputer and back, end-of-partitions are retained.

When a text file is received from a microcomputer, a carriage return is used to indicate the end-of-line. A line feed immediately following the carriage return is ignored.

CYBER Binary Files

A NOS/VE file being transferred to a microcomputer is transferred as a CYBER binary file if the file attributes are `RECORD_TYPE=UNDEFINED`. The file is blocked as it is sent to the microcomputer. The first character of each block indicates the block length. This method allows the exact length of the file to be preserved.

NOS/VE object libraries are transferred in the same manner as CYBER binary files, except that `FILE_STRUCTURE` file attribute is set to `LIBRARY` when the file is received by the host.

You can transfer NOS/VE backup files in the same way you transfer CYBER binary files except that the `RECORD_TYPE` file attribute is set to `VARIABLE` and the `FILE_CONTENTS` attribute is set to `FILE_BACKUP`.

Special Characters

XMODEM uses special file marking characters when transferring CYBER binary files and text files.

- **CYBER binary files**

The character represented by 0F6 hexadecimal indicates end-of-information (EOI).

- **Text files**

Special file marker sequences for text files are explained under Specifying File Markers later in this chapter.

These characters are always inserted into files being transferred from the host to a microcomputer. When these characters are encountered in a file being sent from a microcomputer, they are translated into the appropriate file markers.

To be recognized as a file marker in a CYBER binary file, the file marker characters must appear where the block length indicator is expected. To be recognized as a file marker in a text file, the file marker characters must appear immediately after the end-of-line. If any blocks are received after the EOI character, the blocks are acknowledged with either an ACK or NAK, but the data is ignored.

These characters have no special meaning for a micro binary file.

Setting the High-Order Bit

When a text file is transferred, the high-order bit of each character is not set. If your microcomputer requires this bit to be set, there are two possible solutions.

- You can write a program on the microcomputer to set the high-order bit.
- If the file is going from microcomputer to host to microcomputer and is not being used on the host, you can transfer the file as a micro binary file.

Initiating the File Transfer

The XMODEM command initiates a file transfer. The format of the XMODEM command is:

XMODEM

```
FILE_NAME=file
TRANSFER_DIRECTION=keyword
FILE_TYPE=keyword
LINE_FEED=boolean
SPECIAL_FILE_TYPE_BLOCK=boolean
ERROR_CHECKING=keyword
FILE_MARKERS=keyword
CONFIGURATION_FILE=file
STATUS=status variable
```

None of the parameters for XMODEM are required. You may specify the parameters for the XMODEM command in the usual way as illustrated by the following example.

```
/xmodem file_name=file_1 transfer_direction=receive ..
../file_type=micro_binary line_feed=true ..
../special_file_type_block=false ..
../error_checking=crc file_markers=nos
```


You can also specify parameters using positional notation as shown in the following example:

```
/xmodem file_2 send text
```

Or, you can enter the command name alone. If you enter only the command name, you will be prompted for the `FILE_NAME` and `TRANSFER_DIRECTION` parameters. If `TRANSFER_DIRECTION=RECEIVE`, you will also be prompted for the `FILE_TYPE` parameter. You will not be prompted for the `LINE_FEED`, `SPECIAL_FILE_TYPE_BLOCK`, `ERROR_CHECKING`, `FILE_MARKERS`, `CONFIGURATION_FILE`, or `STATUS` parameters.

Specifying File Name

The `FILE_NAME` (FN) parameter specifies the file to be transferred, and, optionally, how the file is to be positioned. There is no default for this parameter. If you do not specify a file, you will be prompted to do so.

Establishing Transfer Direction

The `TRANSFER_DIRECTION` (TD) parameter indicates the direction of a file transfer. Transfer direction is from the point of view of the host. Specify one of the following options:

SEND

The `SEND` (S) option indicates a file is sent from the host to a microcomputer.

RECEIVE

The `RECEIVE` (R) option indicates a file is received by the host from a microcomputer.

There is no default for the `TRANSFER_DIRECTION` parameter. If you do not make an entry, you will be prompted to do so.

If you specify `SEND`, it indicates the file is sent from the host. Any file sent from the host must have `READ` access and must not be an empty file. If the file you specified on the `FILE_NAME` parameter is empty or cannot be read, you will be prompted for another file name. After you have entered a valid file name and transfer direction, you will receive a message indicating the approximate number of blocks to be transferred. At this point, enter the commands required on your microcomputer to place it in receive mode. The transfer begins when

the microcomputer sends a NAK character (or the letter C for CRC mode) to the host. If you decide not to transfer a file or decide to abort a transfer once it has started, enter CONTROL X (the ASCII CAN character) to end the transfer operation.

If you specify RECEIVE, it indicates the file is received by the host. If the file specified on the FILE_NAME parameter does not exist, the file is created. If the file is one that already contains data, the file must have WRITE access (that is, you must be able to MODIFY, SHORTEN, and APPEND). If the file being received is an old file (that is, a file that has been opened regardless of whether it contains data), the record type must be compatible with the type of file being received. If the file is a CYBER binary or micro binary, the RECORD_TYPE file attribute must be UNDEFINED. If the file is a text file, the RECORD_TYPE file attribute must be VARIABLE.

The following example sends file MAIL from the host to a microcomputer.

```
/xmodem file_name=mail transfer_direction=send
```

To receive file TEST from a microcomputer, enter:

```
/xmodem file_name=test transfer_direction=receive
```

Since the host will be receiving a file, you will be prompted for the FILE_TYPE parameter.

Specifying File Type

The FILE_TYPE (FT) parameter specifies the type of file received by the host. Choose one of the following keyword values:

TEXT (T)

The TEXT option indicates a text file is to be received.

BINARY (B)

The BINARY option indicates a CYBER binary file is to be received.

OBJECT_LIBRARY (OL)

The OBJECT_LIBRARY option indicates an object library is to be received.

MICRO_BINARY (MB)

The **MICRO_BINARY** option indicates a micro binary file is to be received.

VE_BACKUP (V)

The **VE_BACKUP** option indicates that a NOS/VE backup file is to be received.

SELECT_AUTOMATICALLY (SA)

The **SELECT_AUTOMATICALLY** option indicates that the file type to be received will be determined by the first block received.

There is no default for this parameter. If you do not make an entry and **TRANSFER_DIRECTION=RECEIVE**, you will be prompted to do so. This parameter is ignored when **TRANSFER_DIRECTION=SEND**.

After the **FILE_TYPE** parameter is entered, XMODEM checks to ensure that record types are compatible if the file is an old file. If **TEXT**, **BINARY**, **OBJECT_LIBRARY**, or **MICRO_BINARY** are specified and XMODEM determines that record types are incompatible between the file attributes specified and the existing file attributes, you will be prompted for a new **FILE_NAME** and **FILE_TYPE**.

If you have chosen the **SELECT_AUTOMATICALLY** option, XMODEM cannot check for record type compatibility. If you are writing on an old file and you use this option, be sure to check for record compatibility before trying to transfer a file. If record types are not compatible, XMODEM will abort the first time it attempts to write to the old file. For a new file, record compatibility is not a concern. See the **Specifying the File Type Block** section for more information.

After file and record types have been verified, XMODEM issues the following message:

```
Cyber receiving file from micro
```

At this point, you should enter the commands required to place your microcomputer in send mode.

Specifying the End-of-Line

The `LINE_FEED` (LF) parameter specifies whether a line feed is required after a carriage return to signal the end of a line. It is used only for transfers from the host (`TRANSFER_DIRECTION=SEND`). When `TRANSFER_DIRECTION=RECEIVE`, this parameter is ignored and either a carriage return alone or a carriage return and a line feed will indicate the end-of-line.

If `LINE_FEED` is set to `TRUE`, a carriage return and a line feed are transmitted to the microcomputer to indicate the end-of-line. If set to `FALSE`, only a carriage return is transmitted. The default value is `TRUE`.

Specifying the File Type Block

The `SPECIAL_FILE_TYPE_BLOCK` (SFTB) parameter specifies whether a nonstandard block is transmitted. If set to `TRUE`, the nonstandard block is transmitted as the first block of the file. If set to `FALSE`, this block is never transmitted. The default value is `FALSE`.

If this parameter is set to `TRUE`, the first block transferred by XMODEM is of the form

```
81(16) / filetype / 255-filetype / 123D / 128 zeroes
```

where filetype may be:

```
00 - Text
04 - Micro binary
08 - CYBER binary
40(16) - Object library
```

This nonstandard block is recognized by some microcomputer terminal packages (such as ASCII Express³ for the Apple⁴ II microcomputer family). The block is also recognized by the NOS/VE implementation of XMODEM. If this nonstandard block is not recognized by a receiver, the receiver will send an NAK character. After three NAK characters have been sent, XMODEM will proceed with the file transfer.

3. A registered trademark of United Software Industries, Inc.

4. A registered trademark of Apple Computer, Inc.

If the `FILE_TYPE` parameter is set to `SELECT_AUTOMATICALLY` and the first block received is a nonstandard block in the format shown above, `FILE_TYPE` determines the type of transfer that is done. If the file type is not recognized, the file is assumed to be a micro binary file. If the first block received is not in the format shown above when `FILE_TYPE` is set to `SELECT_AUTOMATICALLY`, the file is assumed to be a text file.

Checking for Errors

The `ERROR_CHECKING (EC)` parameter determines whether error detection is done using checksums or using a CRC algorithm. This parameter has effect only when the host is the receiver. If `TRANSFER_DIRECTION=SEND`, the microcomputer determines the error checking method used and this parameter is ignored.

Enter one of the following keyword values:

CRC (CR)

The CRC option indicates that error detection is done using the CCITT cyclic redundancy method.

CHECKSUM (CH)

The CHECKSUM option indicates that error detection is done using the checksum method.

If you do not specify a value for this parameter, CRC is assumed.

The CRC method is the more reliable method for error detection, but is not supported by all implementations for XMODEM. If, after four tries, XMODEM receives no response from the microcomputer, the host will assume that the microcomputer does not support the CRC method and will revert to the checksum method.

Specifying File Markers

The `FILE_MARKERS (FM)` parameter specifies the sequences sent as file markers and what received sequences are interpreted as file markers. This parameter applies only to text files.

Specify one of the following keyword values:

NOS (NOSVE or N)

These keyword values are equivalent. If the FILE_MARKERS parameter is specified as NOS, NOSVE or N, the following file markers are used:

If TRANSFER_DIRECTION=SEND:

#EOR is sent from the host to the microcomputer to indicate end-of-partition.

#EOI is sent from the host to the microcomputer to indicate end-of-information.

If TRANSFER_DIRECTION=RECEIVE:

#EOI indicates end-of-information when received by the host alone on a line. Any characters received after an #EOI are ignored.

#EOR, #EOF, or #EOP causes an end-of-partition to be written when any of the sequences is received by the host alone on a line.

CPM (C or MSDOS or M)

These keyword values are equivalent. If the FILE_MARKERS parameter is specified as CPM, MSDOS or M, the following file markers are used:

If TRANSFER_DIRECTION=SEND:

CONTROL Z is sent by the host to indicate the end-of-information.

End-of-partitions are ignored when sent to a microcomputer.

If TRANSFER_DIRECTION=RECEIVE:

CONTROL Z is recognized by the host as end-of-information.

#EOR, #EOF, and #EOP are recognized by the host as end-of-partition.

If you do not enter a value for this parameter, NOSVE is assumed.

Specifying a Configuration File

The CONFIGURATION_FILE (or CF) parameter allows you to change the default values for the LINE_FEED, SPECIAL_FILE_TYPE_BLOCK, ERROR_CHECKING, and FILE_MARKERS parameters.

The file you specify for this parameter must exist. If you do not specify a file, PFTF uses file \$USER.PFTF_CONFIG as the configuration file. If file \$USER.PFTF_CONFIG does not exist, PFTF assumes there is no configuration file.

Table 14-1 lists the entries that can be supplied within the configuration file.

Table 14-1. Configuration File Entries

Entry Name ¹	Entry Type	Default	Description
LF	boolean	YES	Specifies whether a line feed is required after a carriage return to signal the end of a line. This entry is ignored if TRANSFER_DIRECTION=RECEIVE.
SP	boolean	NO	Specifies whether a nonstandard block is transmitted as the first block of the file.
EC	keyword value (CRC or CHECKSUM)	CRC	Specifies the error detection method used. This entry is ignored if TRANSFER_DIRECTION=SEND.
FM	keyword value (NOSVE, CPM, or MSDOS)	NOSVE	Specifies file markers used.

1. Entry names are constructed so that the same configuration file can be used on NOS/VE or NOS.

The configuration file contains configuration entries you want to use for file transfers. The file is constructed according to the following rules:

- Entries begin in any column.
- One configuration entry per line.
- Spaces are not significant.
- Any line beginning with an asterisk (*) is interpreted as a comment line.
- Each entry may, but is not required to, end with a period.
- If an entry ends with a period, comments may be placed after the period.
- Blank lines are ignored.
- Uppercase and lowercase characters are equivalent.
- Maximum line length is 80 characters.

For example:

```
* This is a sample configuration file.
```

```
lf=no. Send cr instead of cr/lf to micro  
fm=cpm  
ec=checksum
```


KERMIT-VE

KERMIT-VE is the NOS/VE implementation of the KERMIT file transfer protocol.⁵ The protocol involves two programs running on separate computers linked by a communications line. Each program must follow the standard rules which allow them to communicate with each other, regardless of the machine the programs run on.

KERMIT-VE is intended primarily for file transfers between microcomputers and NOS/VE.

Accessing the KERMIT online manual

For detailed information about KERMIT-VE, see the KERMIT online manual. To access the KERMIT online manual, at the NOS/VE system prompt type

```
/help m=kermit
```

or start KERMIT and use the HELP command. To use the KERMIT HELP command, invoke KERMIT on your microcomputer, connect your microcomputer to NOS/VE, log in to NOS/VE, and at the system prompt type

```
/kermit
```

Once you have done this, you will see the KERMIT-VE prompt. At the KERMIT-VE prompt, type

```
KERMIT-VE: help manual
```

and you will see the KERMIT online manual.

NOTE

The NETWORK_COMMAND_CHARACTER attribute should not be the same as the first character of the KERMIT escape sequence (typically the ESC character). For information on how to change the NETWORK_COMMAND_CHARACTER on NOS/VE, see the CHANGE_TERMINAL_ATTRIBUTES command in chapter 13.

5. The KERMIT protocol was developed initially at the Columbia University Center for Computing Activities. The implementation of the KERMIT protocol for NOS/VE was developed with the permission of Columbia University and follows the guidelines for commercial use and distribution of KERMIT as published by Columbia University.

Simple KERMIT File Transfers

Below is an example of how to perform two ASCII file transfers at an IBM Personal Computer (PC) executing MS-DOS Kermit. The PC is connected to a NOS/VE host computer. The IBM PC is local, the NOS/VE system is remote. In the example, FILE1.TXT is sent from the PC to the host file \$USER.SUB_CATALOG_1.FILE1, and then FILE2.TXT is received by the PC from the NOS/VE host file \$USER.SUB_CATALOG_1.FILE2.

NOTE

Some commands in this example are specific to MS-DOS KERMIT. Other versions of KERMIT may use different commands to perform these same tasks.

1. Use the following commands to initiate KERMIT on the PC, set the baud rate to 19200 (note that this assumes a CDCNET connection) and connect to NOS/VE:

```
A>kermit
IBM-PC Kermit-MS V2.28
Type ? for help

Kermit-MS>set baud 19200
Kermit-MS>connect
```

[Connecting to host, type Control-] C to return to PC]

2. At this point you are ready to log in to the host system. Enter whatever commands are necessary to do this. Once you are logged in, you can initiate KERMIT-VE, set your working catalog, and begin receiving a file from the PC by entering the following:

```
/kermit
Welcome to KERMIT-VE, Version 1.0.1

KERMIT-VE:local setwc $user.sub_catalog_1
KERMIT-VE: receive
Receiving...
```

NOTE

The LOCAL command used in the above example allows you to execute any NOS/VE command at the KERMIT-VE prompt.

3. It is necessary at this point to return to the PC side to direct the file transfer. You do this by entering a CTRL/] followed by a C. Once you have returned to the PC side, enter the following to send the file to NOS/VE:

```
Kermit-MS>send file1.txt file1
```

NOTE

The file being sent in the preceding example is sent as FILE1 rather than FILE1.TXT because NOS/VE will not accept a file name with a period (.) in it.

4. The PC displays the progress of the file transfer. When the PC's KERMIT prompt returns to the screen, the transfer is done. When you see the PC's KERMIT prompt, you must connect back to NOS/VE and enter a carriage return to complete the file transfer operation and bring up the KERMIT-VE prompt:

```
Kermit-MS>connect
```

```
KERMIT-VE:
```

NOTE

If you are performing a binary file transfer, you must set the file type on KERMIT-VE to BINARY, and set the parity on both KERMIT-VE and the PC KERMIT to ODD.

5. To send a file from NOS/VE to the PC, enter the KERMIT SEND command, escape to the PC using the control sequence described earlier, and enter the KERMIT RECEIVE command:

```
KERMIT-VE:send file2 file2.txt  
Sending... FILE2 as FILE2.TXT  
Kermit-MS>receive
```

6. Once the transfer is complete, you must return to NOS/VE again to finish the transfer operation by entering a return. You may then quit KERMIT-VE, and log out from NOS/VE.

Desktop/VE

Desktop/VE is an application developed for the Macintosh that extends the Macintosh user interface to Control Data's mainframe computers. Using Desktop/VE and your Macintosh, you can manage mainframe files and applications just as you do Macintosh files -- by using the mouse to manipulate icons.

You can transfer files between your Macintosh and Control Data's mainframes using Desktop/VE. Desktop/VE file transfers are accomplished using the X.PC protocol developed by TYMNET Incorporated.

For more information on using Desktop/VE, see the Desktop/VE for Macintosh usage manual, or the online help that came with your copy of Desktop/VE.

A

Abort

The immediate abnormal termination of a command or task.

Absolute Path

Identifies a file or catalog via a system path, family path, user path, or local path. Contrast with Relative Path.

Alias

Alternative name for a statement or parameter.

Alphabetic Character

One of the following letters:

A through Z

a through z

See also Character and Alphanumeric Character.

Alphanumeric Character

Alphabetic or numeric character. See also Character, Alphabetic Character, and Numeric Character.

American Standard Code for Information Interchange (ASCII)

A 7-bit code representing a prescribed set of characters. NOS/VE stores each 7-bit ASCII code right justified in an 8-bit byte.

ANSI

American National Standards Institute.

ANSI File

A single file whose file data is preceded by an HDR label group (HDR1 and optional HDR2 through HDR9 labels) and followed by an EOF label group (EOF1 and optional EOF2 through EOF9 labels). An ANSI file can span tape volumes.

ANSI_SPANNED Record

See S Type Record.

ANSI_VARIABLE Record

See D Type Record.

Application Value

The kind of value whose syntax and semantics are defined by the application program that interprets the value.

ASCII

See American Standard Code for Information Interchange.

Assignment Statement

A statement that assigns a value to a variable.

Asynchronous Task Execution

A task whose execution is independent of other events. Contrast with Synchronous Task Execution.

B**Batch Mode**

A mode of execution where a job is submitted and processed as a unit with no intervention from the user. Contrast with Interactive Mode.

Beginning-of-Information (BOI)

The file boundary that marks the beginning of the file.

Bit

A binary digit. A bit has a value of 0 or 1. See also Byte.

Block

A logical or physical grouping of data.

BOI

See Beginning-of-Information.

Boolean

A kind of value that is evaluated as TRUE or FALSE.

Boolean Constant

A constant that represents a boolean (logical) value of TRUE or FALSE. One of the following names can be used to specify a boolean constant:

TRUE FALSE

YES NO

ON OFF

Byte

A group of contiguous bits. For NOS/VE, 1 byte is equal to 8 bits. An ASCII character code uses the rightmost 7 bits of 1 byte.

Byte-Addressable File Organization

A file organization in which records are accessed by their byte address.

C**Catalog**

1. A directory of files and catalogs maintained by the system for a user. The catalog \$LOCAL contains only temporary file entries.
2. The part of a path that identifies a particular catalog in a catalog hierarchy. The format is as follows:

name.name.name

where each name is a catalog. See also Catalog Name and Path.

Catalog Name

The name of a catalog in a catalog hierarchy (path). By convention, the name of the user's master catalog is the same as the user's user name.

Character

A letter, digit, space, or symbol that is represented by a code in one or more of the standard character sets.

It is also referred to as a byte when used as a unit of measure to specify block length, record length, and so forth.

A character can be a graphic character or a control character. A graphic character is printable; a control character is nonprintable and is used to control an input or output operation.

Child Job

Job that has been initiated by another job.

Child Task

Task that has been initiated by another task.

COBOL

COmmon Business Oriented Language.

Collating Sequence

The sequence in which characters are ordered for purposes of sorting, merging, and comparing.

Command

A statement that initiates a specific operation on NOS/VE. A command name is recognized by the SCL interpreter if it appears as an entry in the command list.

Command Library

An object library used in the SCL command list.

Command List

One or more entries that define the commands that are currently available. A command list entry is an object library, catalog, or special entry \$SYSTEM.

Command Stream

The source for all statements to be processed by the SCL interpreter.

Command Utility

NOS/VE processor that adds its command table (referred to as its subcommands) to the beginning of the SCL command list. The subcommands are removed from the command list when the processor terminates.

Comment

Any characters or sequence of characters (except the quotation mark) that is preceded by a quotation mark and terminated by another quotation mark or an end of line. A comment is treated exactly as a space.

Compiler

A processor that accepts source code as input and generates object code as output.

Condition Code

Alphanumeric characters that uniquely identify a NOS/VE diagnostic. The condition code is returned as part of the status record when an abnormal status occurs.

Condition Handler

A WHEN statement to which control is transferred when a condition occurs. The WHEN statement is executed only if it has been established as the condition handler for the specified condition and if the condition occurs in its scope.

Condition Name

A string value that corresponds to a specified condition code. Condition codes and names are used in status processing.

Connection Attribute

The characteristics that relate to a terminal connection. A terminal has a separate set of connection attributes for each connection. See Terminal Attribute.

Control Command

A system command that you cannot remove, although you can override it. An example of a control command is LOGIN.

Control Family

The family name of the control user.

Control Statement

A statement used to structure and control the flow of a job.

Control User

The user name under which a user submits a job. With the following exceptions, the control user is the same as the login user.

- Jobs entered via the SUBMIT_JOB command, DETACH_JOB command, or JOB/JOBEND command pair inherit the parent job's control user.
- Jobs entered from a private I/O station have the private station operator as the control user for the job.

cpu

Characters per inch, a measure of the tape recording density.

CYBIL

CYBER Implementation Language. The implementation language of NOS/VE.

Cycle

A numbered version of a file that can be registered with the same file name and in the same catalog as the other versions of the file. Each permanent file can have from 1 through 999 cycles.

See also Cycle Number and Cycle Reference.

Cycle Number

An unsigned integer from 1 through 999 that identifies a specific version of a permanent file.

Cycle Reference

The cycle of a permanent file to be accessed. A cycle reference can be either an unsigned integer or one of the following designators:

\$HIGH

\$LOW

\$NEXT

D

D Type Record

Variable length records as defined by the ANSI Standard.

Default

The assumed value for a parameter when the parameter is not specified by the user.

Delimiter

The indicator that separates and organizes data.

Digit

One of the following characters:

0 1 2 3 4 5 6 7 8 9

Direct-Access File Organization

A keyed-file organization in which each record is accessed directly by hashing its primary-key value. Records can be accessed sequentially, but the records are not returned in sorted order. Contrast with Indexed-Sequential File Organization.

Directive

A statement that specifies processing of a command or subcommand.

Display Code

A 64-character subset of the ASCII code, which consists of alphabetic letters, symbols, and numerals. This character set is not used by NOS/VE.

Dual State

The state in which two operating systems execute simultaneously on the same mainframe. NOS/VE and either NOS Version 2 or NOS/BE Version 1.5 are such partners.

E**Ellipsis**

1. Two or more consecutive periods at the end of a physical line to indicate command line continuation. The ellipsis can be optionally preceded and/or followed by a space.
2. Two consecutive periods separating two values to indicate a range of values in a parameter list. See also Value Element.

Embedded Key

A key that is contained within a record, as opposed to a key that is defined outside a record (such as in the Working-Storage or Common-Storage sections of a COBOL program).

End-of-Information (EOI)

The point at which data in the file ends.

End-of-Partition (EOP)

A special delimiter in a file with variable record type.

Environment Objects

Elements that make up an SCL command environment. These elements include such things as file connections, program attributes and the command list.

EOI

See End-of-Information.

EOP

See End-of-Partition.

Epilog

The SCL statement list that is executed at the end of a job.

Execution Ring

The level of hardware privilege assigned to a procedure while it is executing.

Expression

Notation that represents a value. A constant or variable appearing alone, or combinations of constants, variables, and operators.

F**F Type Record**

Fixed-length records, as defined by the ANSI Standard.

Family

A logical grouping of NOS/VE users that determines the location of their permanent files.

Family Administrator

The family member who creates, deletes, and otherwise manages the validations of other members of the family. A user with system administration capability assigns a family administrator at the time the family is created.

Family Name

A name that identifies a NOS/VE family. See also Family.

Family Path

Identifies a file via a family name and a user path as follows:

:family name.user path

or

\$FAMILY.user path

Field

A subdivision of a record that is referenced by name. For example, the field NORMAL in a record of type OST\$STATUS called OLD_STATUS is referenced as follows:

OLD_STATUS.NORMAL

File

1. A collection of information referenced by a name.
2. An SCL element that specifies a temporary or permanent file, including its path and, optionally, a cycle reference (for permanent files). A file is identified by specifying a path and, optionally, a cycle reference (for permanent files) as follows:

path.cycle reference

File Attribute

A characteristic of a file. Each file has a set of attributes that defines the file structure and processing limitations.

File Connection

State where any data access requests that are made for a file are passed on to a second file as well.

File Name

The name of a NOS/VE file. It is used in a file reference to identify the file. See also Name.

File Organization

Defines the way records are stored in a file. The available file organizations are sequential, byte-addressable, direct-access, and indexed-sequential.

File Position

The location in the file at which the next read or write operation will begin. A file that can be positioned is identified by specifying a path, an optional cycle reference (for permanent files), and an optional file position as follows:

path.cycle reference.file position

The file position designators are:

\$ASIS Leave the file in its current position.

\$BOI Position the file at the beginning-of-information.

\$EOI Position the file at the end-of-information.

See also Path and Cycle Reference.

File Reference

An SCL element that identifies a file and, optionally, the file position established prior to the file's use. The format of a file reference is:

:family.catalog.file.cycle.file position

where catalog is one or more catalog names separated by a period.

where file is a 1- to 31-character name.

where cycle is a numeric value from 1 to 999 that represents a version of the file.

where the file positions are:

\$BOI

\$ASIS

\$EOI

See also File and File Position.

FORTRAN

FORmula TRANslating system.

Function

A statement that performs a specific action and can be called by name from a statement elsewhere in a program. Normally, a function computes a value and returns it to the portion of the program that called it. An example is the \$DATE function, which computes the current date in one of a number of formats.

H

Hashing Procedure

The CYBIL procedure used to relate a primary-key to a home block number in a direct-access file.

Home Block

A unit of space in a direct-access file.

I

Indexed-Sequential File Organization

A keyed-file organization in which each record is accessed by finding its primary-key value in the file index. When records are read sequentially from an indexed-sequential file, they are returned sorted by primary-key value. Contrast with Direct-Access File Organization.

Input Lines

One or more SCL statements. Statements on a single line are separated by semicolons. Input lines can be continued on more than one physical line by placing an ellipsis at the end of each physical line to be continued. Input lines have a maximum length of 65,535 characters.

Integer

A value representing one of the numbers 0, +1, -1, +2, -2, and so forth.

Integer Constant

A constant that represents an integer value. One or more digits and, for hexadecimal integer constants, the following characters:

A B C D E F a b c d e f

A hexadecimal integer constant must begin with a digit. A preceding sign and subsequent radix are optional.

Interaction Style

Method by which users enter and receive data on your terminal. The interaction style may be either line or screen mode.

Interactive Mode

A mode of execution where the user enters commands at a work station and each command elicits a response from the host. Contrast with Batch Mode.

J

Job

A set of tasks executed for a user name. NOS/VE accepts interactive and batch jobs. In interactive mode, a job is usually the same as a terminal session.

Job Attribute

A characteristic of a job.

Job Class

Name that defines a set of attributes assigned to a job. These attributes control the operation of the job during its input and initiation phases. For instance, the job class determines when a particular job is initiated. The default job classes used by NOS/VE are SYSTEM, MAINTENANCE, BATCH, INTERACTIVE, and UNASSIGNED.

Job File

A file that contains the statements and input data for the job and the output produced by the job. The job files are identified by the following names:

```
COMMAND    OUTPUT
INPUT      $NULL
$JOB_LOG
```

Job Limits

Limits that are used to define the amount of resources a job may use.

Job Log

A chronological listing that receives error messages and information messages from commands.

K**Key**

A string of characters or a number that the user defines to uniquely identify a record.

Keyed File

A type of file that provides for record access by a primary-key value. Currently, NOS/VE supports two types of keyed files: indexed sequential and direct access.

Keyword

A parameter value that has special meaning in the context of a particular parameter. For example, a parameter called COUNT might normally expect an integer but could be given the keyword ALL.

L**Label**

A name used to reference a structured statement.

Line Mode

Method of interacting with a utility a line at a time.

Line Mode Interaction

Interaction with NOS/VE is performed on a line by line basis. The user enters a single line of data and waits for NOS/VE to respond before entering another line of data.

Link Attributes

Attributes which are used to specify login information to the partner side of a dual-state system.

List

A command format notation specifying that a parameter can be given several values.

Load Module

A module reformatted for code sharing and efficient loading. When the user generates an object library, each object module in the module list is reformatted and written as a load module on the object library.

Local File

A file that is accessed via the local catalog named \$LOCAL. See also File, Path, and Local Path.

Local Path

Identifies a local file as follows:

\$LOCAL.file name

Local System

The operating system your interactive job's or a batch job's parent job is currently executing on.

Lock

A mechanism that makes a primary-key value (or, for a file lock, all primary-key values) inaccessible to other instances of open of the file.

Login User

The user name under which a job is scheduled and run. For batch jobs, the user name appearing in the USER parameter of the LOGIN command is the login user.

M**Mainframe Attribute**

A characteristic of the mainframe on which the NOS/VE operating system is running.

Mass Storage

Disk storage that allows random file access and permanent file storage.

Mass Storage File

A file stored on a disk. The file can have any organization.

Master Catalog

The catalog the system creates for each user name. The master catalog contains entries for all permanent files and catalogs a user creates. The name of the master catalog is the same as the user name.

Module

A unit of text accepted as input by the loader, linker, or object library generator. See also Object Module and Load Module.

Multifile Set

A set of more than one ANSI file residing on one or more tape volumes.

N**NAM**

See Network Access Method.

Name, SCL

Combination of from 1 through 31 characters chosen from the following set:

- Alphabetic characters (A through Z and a through z).
- Digits (0 through 9).
- Special characters: #, @, \$, -, [,], \, ^, ` , {, }, |, ~.

The first character of a name cannot be numeric.

Natural Language

Language used for messages and help information produced in a job.

NCC

See Network Command Character.

Network Access Method (NAM)

A NOS software package that provides communication between terminal users and the host computer.

Network Command Character

Character which indicates that the data appended to the character is to be handled in a special way by the network.

Nonembedded Key

A primary key that is not physically contained in the record. Contrast with Embedded Key.

NOS

Acronym for Network Operating System, an operating system for the host computer. NOS controls the computation of programs submitted through remote terminals and maintains normal batch processing operations for jobs submitted locally.

NOS/BE

Acronym for Network Operating System/Batch Environment, an operating system for the host computer. NOS/BE controls the computation of programs submitted through remote terminals and maintains normal batch processing operations for jobs submitted locally.

NOS/VE

Acronym for Network Operating System/Virtual Environment, an operating system for the mainframe computer.

NOS/VE has all of the capabilities of NOS. In addition, NOS/VE uses virtual memory.

NTF System

Any remote system that is connected to the local system through the Network Transfer Facility (NTF) queued file transfer application. For more information on NTF, see your system administrator.

Numeric Character

Any digit from 0 through 9.

O**Object Code**

Executable code produced by a compiler.

Object File

A file containing one or more object modules.

Object Library

A file containing one or more load, SCL procedures, program description, message, and/or load modules and a dictionary to each module.

Object Module

A compiler-generated unit containing object code and instructions for loading the object code. It is accepted as input by the system loader and the CREATE_OBJECT_LIBRARY utility.

Open Operation

A set of preparatory operations performed on a file before file input and output can take place.

Operand

An entity to which an operation is applied.

Operator

The symbol that represents the action to be performed in an operation.

P**Padding**

Space deliberately left unused. Fixed-length records are padded if the data provided for the record is shorter than the record length. Within keyed files, blocks are padded during file creation to allow easy addition or expansion during later file updates.

Parameter

A value list optionally preceded by and equated to a parameter name. For example:

parameter name = value list

or

value list

Parameter Definition

Specifies the parameter names, value specification, and default specification for an SCL procedure parameter.

Parameter Expression

An expression that, when evaluated, results in a parameter value.

Parameter List

A series of parameters separated by spaces or commas.

Parameter Name

A name that uniquely identifies a parameter.

Parameter Value

See Value.

Parent Job

A job that initiates another job.

Parent Task

Task that initiates another task.

Partner System

The operating system sharing a dual-state mainframe with the local system. See Local System.

Path

In NOS/VE, a path specifies the location of a file or catalog in a catalog hierarchy. A general example of a path, from highest to lowest level in its hierarchy, is family name, user name (or master catalog name), subcatalog name(s), and file name.

Pause Break

Action which allows you to suspend an executing command so that you can perform other activities, such as checking the status of your job or consulting an online manual. Also called user break 1.

Permanent Catalog

A catalog of permanent files.

Permanent File

A mass storage file preserved by NOS/VE across job executions and system deadstarts. A permanent file has an entry in a permanent catalog. See also File.

Position-Dependent Parameter

A parameter that must appear in a specified location, relative to other parameters. Contrast with Position-Independent Parameter.

Position-Independent Parameter

A parameter that consists of a parameter name followed by a value list. Contrast with Position-Dependent Parameter.

Primary Key

The required record key in a keyed file. The primary-key value must be defined for a file when the file is first created, and each record in the file must have a unique value for the key.

Primary Task

Task to which break conditions (such as pause break) are sent.

Private Reader

Task that does not share its current file position with any other current accessors of that file.

Procedure

A sequence of SCL commands executed when the procedure name is entered. It can be stored as a module on an object library.

Procedure Library

An object library that contains SCL procedures.

Processor Attribute

A characteristic of the hardware processor on which the system is running.

Program Attribute

A characteristic of a program as defined in the program description or by a job default value.

Program Description

Information that defines a program, including the modules that comprise the program and the options to be used when it is executed.

Prolog

The SCL statement list that is executed at the beginning of each job.

Q

QTF System

Any remote system that is connected to the local system through the QTF queue file transfer application. For more information on QTF, see the RHF Usage manual.

R

Radix

Specifies the base of a number. NOS/VE recognizes number bases from base 2 through base 16. A radix enclosed in parentheses must follow a nondecimal number.

See also Integer Constant.

Random Access

The process of reading or writing a record in a file without having to read or write the preceding records; applies only to mass storage files. Contrast with Sequential Access.

Range

Value represented as two values separated by an ellipsis. The element is associated with the values from the first value through the second value. The first value must be less than or equal to the second value. For example:

value..value

Real Number

A number that has a decimal point and at least one digit on each side of the decimal point.

Relative Path

Identifies a file via defaults established with the current working catalog or an absolute path. A relative path is used in a family path, user path, and local path. SCL supplies any omitted values necessary to create an absolute path.

Remote System

An operating system that resides on a mainframe other than the one your interactive job's or a batch job's parent job is currently executing on.

Remote Validation

Condition that occurs when the necessary information has been provided to NOS/VE that allows a user to access files residing on a remote system.

Ring

Level of hardware protection given a file or segment. A file is protected from unauthorized access by tasks executing in higher rings. See also Execution Ring.

Ring Attributes

A file attribute whose value consists of three ring numbers, referred to as r1, r2, and r3. The ring numbers define the four ring brackets for the file as follows:

Read bracket is 1 through r2.

Write bracket is 1 through r1.

Execute bracket is r1 through r2.

Call bracket is r2+1 through r3.

S**S Type Record**

Spanned records as defined by the ANSI Standard.

SCL

See System Command Language.

Screen Mode

Method of interacting with a utility a screen at a time.

Screen Mode Interaction

Data is entered and displayed in different areas of the screen.

SCU

See Source Code Utility.

Sequential Access

The processing of records in order (physical or logical). Contrast with Random Access.

Sequential File Organization

A file with records stored and retrieved in the order in which they were written. No logical order exists other than the relative physical record position.

Service

Program running on a computer that handles the flow of information between the network and the computer.

Service Class

Defines characteristics that govern the execution phase of job processing.

Sign

Indicates whether a number is positive or negative. It is one of the following characters:

+	Positive number
-	Negative number

If a sign is omitted, the number is positive.

See also Integer Constant.

Site Administrator

The family member who creates, deletes, and otherwise manages the other members of the family. A system analyst assigns a site administrator at the time the family is created.

Source Code

Statements written for input to a compiler.

Source Code Utility (SCU)

A NOS/VE command utility that stores, organizes, manipulates, and extracts units of text. It is a development tool for large systems or application development groups.

Source Library

A collection of decks on a file, with a header describing the collection, generated and manipulated by the Source Code Utility (SCU).

SRUs

See System Resource Units.

Standard File

A file that provides a default file for use by job files and other files. The standard files are identified by the following names:

\$ECHO

\$ERRORS

\$INPUT

\$LIST

\$OUTPUT

\$RESPONSE

Standard Output

Output produced at the end of a batch job. The output contains the job log and any information written to standard file OUTPUT.

Statement

The basic unit of input that is interpreted by SCL. The following are the SCL statement types.

Assignment statement

Command

Control command

Control statement

Statement List

One or more statements separated by delimiters.

Statically Linked Block

Block defined directly within another block that has access to the outer block's local variables.

Station

A collection of batch I/O devices.

Status Variable

A variable record of kind status that holds the completion status of a command.

String

A value that represents a sequence of characters.

String Constant

A sequence of characters delimited by apostrophes ('). An apostrophe can be included in the string by specifying two consecutive apostrophes.

String Length

An integer that specifies the length of a string.

Structured Statement

A control statement used to structure a job into groups of statements that are processed as a separate entity or until a specified termination condition or statement is encountered.

Subcatalog

A catalog registered in the master catalog or another subcatalog. See also Catalog.

Subcommand

A command that has been added to the command list by a command utility.

Synchronous Task Execution

A task whose execution is dependent upon the occurrence of another event. Contrast with Asynchronous Task Execution.

\$SYSTEM Catalog

The catalog containing all the NOS/VE operating system and accompanying product files.

System Command Language (SCL)

The block-structured interpretive language that provides an interface to the features and capabilities of NOS/VE. All commands and statements are interpreted by SCL before being processed by the system.

\$SYSTEM Command List Entry

The list of all commands provided by NOS/VE.

System File

See Job File and Standard File.

System Operator

A user with special privileges. These privileges are available only at the system console.

System Path

Identifies a file or catalog in the system catalog. Its format is as follows:

\$SYSTEM.relative path

System Resource Units (SRUs)

An accounting unit used to measure resource usage by a job or task.

System-Supplied Job Name

Unique, 19-character name the system gives each job you submit.

T**Tape Volume**

Single reel of magnetic tape used for mass storage of file data.

Task

The instance of execution of a program.

Terminal Attribute

The physical characteristics that relate to a terminal. These attributes apply to all connections from the terminal. See also Connection Attribute.

Terminal Class

Interactive terminal type; the system associates a set of default terminal attributes with the terminal type.

Terminal Definition File

The source file used in defining a terminal for use with a full-screen application.

Terminate Break

Action which allows you to terminate an executing command. Also called user break 2.

U

U-Type Record

A record for which the record structure is undefined.

User Break 1

See Pause Break.

User Break 2

See Terminate Break.

User Name

A name that identifies a NOS/VE user and the location of a user's permanent files in the user's family.

User Path

Identifies a file or catalog via a user name and, optionally, a relative path as follows:

.user name.relative path

or

\$USER.relative path

Utility

See Command Utility.

V

V-Type Record

Variable-sized record; system default record type. Each V-type record has a record header. The header contains the record length and the length of the preceding record.

Value

An expression or application value specified in a parameter list. Each value must match the defined kind of value for the parameter. Keywords, constants, and variable references are all values.

Value Count

An integer expression indicating the number of value elements supplied for a parameter.

Value Element

A single value or a range of values represented by two values separated by an ellipsis. For example:

value

or

value..value

See also Value, Value List, and Value Set.

Value List

A series of value sets separated by spaces or commas and enclosed in parentheses. If only one value set is given in the list, the parentheses can be omitted. For example:

(value set,value set,value set)

or

value set

See also Value, Value Element, and Value Set.

Value Set

A series of value elements separated by spaces or commas and enclosed in parentheses. If only one value element is given in the set, the parentheses can be omitted. For example:

(value element,value element,value element)

or

value element

See also Value, Value Element, and Value List.

Value Set Count

An integer expression indicating the number of value sets supplied for a parameter.

Variable

1. Represents a data value.
2. SCL defines the following kinds of variables:

string

integer

real

boolean

status

Variable Name

A name that identifies a variable.

Variable Reference

An integer, string, or boolean variable reference identifying an integer, string, or boolean variable by its name and optional subscript. For example:

variable name(subscript)

A status variable reference identifies a status variable by its name and optional subscript and/or field. For example:

variable name(subscript).field

W

Working Catalog

The catalog prefixed to a file reference that begins with a name (that is, not a period, colon, or a system name beginning with a dollar sign). The working catalog is the catalog used if no other catalog is specified on a file reference.

Related Manuals

B

The following lists the categories of manuals which relate to NOS/VE.

Ordering Printed Manuals	B-1
Accessing Online Manuals	B-1
Table B-1. Related Manuals	B-2
NOS/VE Site Manuals	B-2
NOS/VE User Manuals	B-3
CYBIL Manuals	B-5
FORTRAN Manuals	B-6
COBOL Manuals	B-7
Other Compiler Manuals	B-7
VX/VE Manuals	B-8
Data Management Manuals	B-10
Information Management Manuals	B-11
CDCNET Manuals	B-11
Miscellaneous Manuals	B-12
Hardware Manuals	B-14

If you are familiar with the SCL System Interface, SCL Language Definition, and SCL Quick Reference manuals, you will find they are retitled and reorganized for NOS/VE release 1.3.1, PSR level 700.

Descriptions of the changes follow:

SCL System Interface and SCL Language Definition

The SCL System Interface and SCL Language Definition manuals are replaced by a single manual, *NOS/VE System Usage*. NOS/VE System Usage contains the information you once found in the two manuals, except for the formats of commands and functions. Look for the command and function formats in the NOS/VE Commands and Functions manual.

SCL Quick Reference

The SCL Quick Reference manual is retitled *NOS/VE Commands and Functions*. It contains the same information, but is organized differently. Book 1 describes the formats of the commands and functions not associated with utilities. Book 2 describes the commands and subcommands of the command utilities.

All NOS/VE manuals and related hardware manuals are listed in table B-1. If your site has installed the online manuals, you can find an abstract for each NOS/VE manual in the online System Information manual. To access this manual, enter:

```
/explain
```

Ordering Printed Manuals

To order a printed Control Data manual, send an order form to:

Control Data Corporation
Literature and Distribution Services
308 North Dale Street
St. Paul, Minnesota 55103

To obtain an order form or to get more information about ordering Control Data manuals, write to the above address or call (612) 292-2101. If you are a Control Data employee, call (612) 292-2100.

Accessing Online Manuals

To access the online version of a printed manual, log in to NOS/VE and enter the online title on the EXPLAIN command (table B-1 supplies the online titles). For example, to see the NOS/VE Commands and Functions manual, enter:

```
/help manual=sc1
```

The examples in some printed manuals exist also in the online Examples manual. To access this manual, enter:

```
/help manual=examples
```

When EXAMPLES is listed in the Online Manuals column in table B-1, that manual is represented in the online Examples manual.

Table B-1. Related Manuals

Manual Title	Publication Number	Online Manuals¹
NOS/VE Site Manuals:		
CYBER 930 Computer System Guide to Operations Usage	60469560	
CYBER Initialization Package (CIP) Reference Manual	60457180	
Desktop/VE Host Utilities Usage	60463918	
MAINTAIN_MAIL ² Usage		MAIM
NOS/VE Accounting Analysis System Usage	60463923	
NOS/VE Accounting and Validation Utilities for Dual State Usage	60458910	
NOS/VE LCN Configuration and Network Management Usage	60463917	
NOS/VE Network Management Usage	60463916	
NOS/VE Operations Usage	60463914	

1. This column lists the title of the online version of the manual and indicates whether the examples in the printed manual are in the online Examples manual.

2. To access this manual, you must be the administrator for MAIL/VE.

(Continued)

Table B-1. Related Manuals (Continued)

Manual Title	Publication Number	Online Manuals¹
Site Manuals (Continued):		
NOS/VE System Performance and Maintenance Volume 1: Performance Usage	60463915	
NOS/VE System Performance and Maintenance Volume 2: Maintenance Usage	60463925	
NOS/VE User Validation Usage	60464513	
NOS/VE User Manuals:		
EDIT_CATALOG Usage		EDIT_ CATALOG
EDIT_CATALOG for NOS/VE Summary	60487719	
Introduction to NOS/VE Tutorial	60464012	
NOS/VE Advanced File Management Tutorial	60486412	AFM_T

1. This column lists the title of the online version of the manual and indicates whether the examples in the printed manual are in the online Examples manual.

(Continued)

Table B-1. Related Manuals (Continued)

Manual Title	Publication Number	Online Manuals¹
NOS/VE User Manuals (Continued):		
NOS/VE Advanced File Management Usage	60486413	AFM
NOS/VE Advanced File Management Summary	60486419	
NOS/VE Commands and Functions Quick Reference	60464018	SCL
NOS/VE File Editor Tutorial/Usage	60464015	EXAMPLES
NOS/VE Object Code Management Usage	60464413	OCM
NOS/VE Screen Formatting Usage	60488813	EXAMPLES
NOS/VE Source Code Management Usage	60464313	SCM and EXAMPLES
NOS/VE System Usage	60464014	EXAMPLES
NOS/VE Terminal Definition Usage	60464016	
Screen Design Facility for NOS/VE Usage	60488613	SDF

1. This column lists the title of the online version of the manual and indicates whether the examples in the printed manual are in the online Examples manual.

(Continued)

Table B-1. Related Manuals (Continued)

Manual Title	Publication Number	Online Manuals¹
CYBIL Manuals:		
CYBIL for NOS/VE File Management Usage	60464114	EXAMPLES
CYBIL for NOS/VE Keyed-File and Sort/Merge Interfaces Usage	60464117	EXAMPLES
CYBIL for NOS/VE Language Definition Usage	60464113	CYBIL and EXAMPLES
CYBIL for NOS/VE Sequential and Byte-Addressable Files Usage	60464116	EXAMPLES
CYBIL for NOS/VE System Interface Usage	60464115	EXAMPLES

1. This column lists the title of the online version of the manual and indicates whether the examples in the printed manual are in the online Examples manual.

(Continued)

Table B-1. Related Manuals (Continued)

Manual Title	Publication Number	Online Manuals ¹
FORTRAN Manuals:		
FORTRAN Version 1 for NOS/VE Language Definition Usage	60485913	EXAMPLES
FORTRAN Version 1 for NOS/VE Quick Reference		FORTRAN
FORTRAN Version 2 for NOS/VE Language Definition Usage	60487113	EXAMPLES
FORTRAN Version 2 for NOS/VE Quick Reference		VFORTRAN
FORTRAN for NOS/VE Tutorial	60485912	FORTRAN_T
FORTRAN for NOS/VE Topics for FORTRAN Programmers Usage	60485916	
FORTRAN for NOS/VE Summary	60485919	
COBOL Manuals:		
COBOL for NOS/VE Summary	60486019	

1. This column lists the title of the online version of the manual and indicates whether the examples in the printed manual are in the online Examples manual.

(Continued)

Table B-1. Related Manuals (Continued)

Manual Title	Publication Number	Online Manuals¹
COBOL Manuals (Continued):		
COBOL for NOS/VE Tutorial	60486012	COBOL_T
COBOL for NOS/VE Usage	60486013	COBOL and EXAMPLES
Other Compiler Manuals:		
ADA for NOS/VE Usage	60498113	ADA
ADA for NOS/VE Reference Manual	60498118	EXAMPLES
APL for NOS/VE File Utilities Usage	60485814	
APL for NOS/VE Language Definition Usage	60485813	
BASIC for NOS/VE Summary Card	60486319	
BASIC for NOS/VE Usage	60486313	BASIC
LISP for NOS/VE Usage Supplement	60486213	
Pascal for NOS/VE Summary Card	60485619	

1. This column lists the title of the online version of the manual and indicates whether the examples in the printed manual are in the online Examples manual.

(Continued)

Table B-1. Related Manuals (Continued)

Manual Title	Publication Number	Online Manuals ¹
Other Compiler Manuals (Continued):		
Pascal for NOS/VE Usage	60485613	PASCAL and EXAMPLES
Prolog for NOS/VE Quick Reference	60486718	PROLOG
Prolog for NOS/VE Usage	60486713	
VX/VE Manuals:		
C/VE for NOS/VE Quick Reference		C
C/VE for NOS/VE Usage	60469830	
DWB/VX Introduction and User Reference Tutorial/Usage	60469890	
DWB/VX Macro Packages Guide Usage	60469910	
DWB/VX Preprocessors Guide Usage	60469920	
DWB/VX Text Formatters Guide Usage	60469900	

1. This column lists the title of the online version of the manual and indicates whether the examples in the printed manual are in the online Examples manual.

(Continued)

Table B-1. Related Manuals (Continued)

Manual Title	Publication Number	Online Manuals¹
VX/VE Manuals (Continued):		
VX/VE Administrator Guide and Reference Tutorial/Usage	60469770	
VX/VE An Introduction for UNIX Users Tutorial/Usage	60469980	
VX/VE Programmer Guide Tutorial	60469790	
VX/VE Programmer Reference Usage	60469820	
VX/VE Support Tools Guide Tutorial	60469800	
VX/VE User Guide Tutorial	60469780	
VX/VE User Reference Usage	60469810	

1. This column lists the title of the online version of the manual and indicates whether the examples in the printed manual are in the online Examples manual.

(Continued)

Table B-1. Related Manuals (Continued)

Manual Title	Publication Number	Online Manuals¹
Data Management Manuals:		
DM Command Procedures Reference Manual	60487905	
DM Concepts and Facilities Manual	60487900	
DM Error Message Summary for DM on CDC NOS/VE	60487906	
DM Fundamental Query and Manipulation Manual	60487903	
DM Report Writer Reference Manual	60487904	
DM System Administrator's Reference Manual for DM on CDC NOS/VE	60487902	
DM Utilities Reference Manual for DM on CDC NOS/VE	60487901	

1. This column lists the title of the online version of the manual and indicates whether the examples in the printed manual are in the online Examples manual.

(Continued)

Table B-1. Related Manuals (Continued)

Manual Title	Publication Number	Online Manuals¹
Information Management Manuals:		
IM/Control for NOS/VE Quick Reference	L60488918	CONTROL
IM/Control for NOS/VE Usage	60488913	
IM/Quick for NOS/VE Tutorial	60485712	
IM/Quick for NOS/VE Summary	60485714	
IM/Quick for NOS/VE Usage		QUICK
CDCNET Manuals:		
CDCNET Access Guide	60463830	CDCNET_ACCESS
CDCNET Batch Device User Guide	60463863	CDCNET_BATCH
CDCNET Commands Quick Reference	60000020	
CDCNET Configuration and Site Administration Guide	60461550	
CDCNET Diagnostic Messages	60461600	
CDCNET Conceptual Overview	60461540	

1. This column lists the title of the online version of the manual and indicates whether the examples in the printed manual are in the online Examples manual.

(Continued)

Table B-1. Related Manuals (Continued)

Manual Title	Publication Number	Online Manuals¹
CDCNET Manuals (Continued):		
CDCNET Network Analysis	60461590	
CDCNET Network Configuration Utility		NETCU
CDCNET Network Configuration Utility Summary Card	60000269	
CDCNET Network Operations	60461520	
CDCNET Network Performance Analyzer	60461510	
CDCNET Product Descriptions	60460590	
CDCNET Systems Programmer's Reference Manual Volume 1 Base System Software	60462410	
CDCNET Systems Programmer's Reference Manual Volume 2 Network Management Entities and Layer Interfaces	60462420	
CDCNET Systems Programmer's Reference Manual Volume 3 Network Protocols	60462430	
CDCNET Terminal Interface Usage	60463850	
CDCNET TCP/IP Usage	60000214	

1. This column lists the title of the online version of the manual and indicates whether the examples in the printed manual are in the online Examples manual.

(Continued)

Table B-1. Related Manuals (Continued)

Manual Title	Publication Number	Online Manuals¹
Migration Manuals:		
Migration from IBM to NOS/VE Tutorial/Usage	60489507	
Migration from NOS to NOS/VE Tutorial/Usage	60489503	
Migration from NOS to NOS/VE Standalone Tutorial/Usage	60489504	
Migration from NOS/BE to NOS/VE Tutorial/Usage	60489505	
Migration from NOS/BE to NOS/VE Standalone Tutorial/Usage	60489506	
Migration from VAX/VMS to NOS/VE Tutorial/Usage	60489508	
Miscellaneous Manuals:		
Applications Directory	60455370	
CONTEXT Summary Card	60488419	
CYBER Online Text for NOS/VE Usage	60488403	CONTEXT
Control Data CONNECT User's Guide	60462560	

1. This column lists the title of the online version of the manual and indicates whether the examples in the printed manual are in the online Examples manual.

(Continued)

Table B-1. Related Manuals (Continued)

Manual Title	Publication Number	Online Manuals¹
Miscellaneous Manuals (Continued):		
Debug for NOS/VE Quick Reference		DEBUG
Debug for NOS/VE Usage	60488213	
Desktop/VE for Macintosh Tutorial	60464502	
Desktop/VE for Macintosh Usage	60464503	
NOS/VE Diagnostic Messages Usage	60464613	MESSAGES
MAIL/VE Summary Card	60464519	
MAIL/VE Usage		MAIL_VE
Math Library for NOS/VE Usage	60486513	
NOS/VE Examples Usage		EXAMPLES
NOS/VE System Information		NOS_VE

1. This column lists the title of the online version of the manual and indicates whether the examples in the printed manual are in the online Examples manual.

(Continued)

Table B-1. Related Manuals (Continued)

Manual Title	Publication Number	Online Manuals¹
Miscellaneous Manuals (Continued):		
Programming Environment for NOS/VE Usage		ENVIRON- MENT
Programming Environment for NOS/VE Summary	60486819	
Professional Programming Environment for NOS/VE Quick Reference		PPE
Professional Programming Environment for NOS/VE Usage	60486613	
Remote Host Facility Usage	60460620	
Hardware Manuals:		
CYBER 170 Computer Systems Models 825, 835, and 855 General Description Hardware Reference	60459960	
CYBER 170 Computer Systems, Models 815, 825, 835, 845, and 855 CYBER 180 Models 810, 830, 835, 840, 845, 850, 855, and 860 Codes Booklet	60458100	

1. This column lists the title of the online version of the manual and indicates whether the examples in the printed manual are in the online Examples manual.

(Continued)

Table B-1. Related Manuals (Continued)

Manual Title	Publication Number	Online Manuals ¹
Hardware Manuals (Continued):		
CYBER 170 Computer Systems, Models 815, 825, 835, 845, and 855 CYBER 180 Models 810, 830, 835, 840, 845, 850, 855, and 860 Maintenance Register Codes Booklet	60458110	
HPA/VE Reference	60461930	
Virtual State Volume II Hardware Reference	60458890	
7021-31/32 Advanced Tape Subsystem Reference	60449600	
7221-1 Intelligent Small Magnetic Tape Subsystem Reference	60461090	

1. This column lists the title of the online version of the manual and indicates whether the examples in the printed manual are in the online Examples manual.

Character Set

C

ASCII Character Set	C-1
EBCDIC Character Set	C-5

ASCII Character Set

This appendix lists the ASCII character set (refer to table C-1).

NOS/VE supports the American National Standards Institute (ANSI) standard ASCII character set (ANSI X3.4-1977). NOS/VE represents each 7-bit ASCII code in an 8-bit byte. These 7 bits are right justified in each byte. For ASCII characters, the eighth or leftmost bit is always zero. However, in NOS/VE the leftmost bit can also be used to define an additional 128 characters.

If you want to define additional non-ASCII characters, be certain that the leftmost bit is available in your current working environment. The full screen applications (such as the EDIT_FILE utility, the EDIT_CATALOG utility, and the programming language environments) already use this bit for special purposes. Therefore, these applications accept only the standard ASCII characters. In applications in which the leftmost bit is not used, however, you are free to use it to define the interpretation of each character as you wish.

ASCII Character Set

Table C-1. ASCII Character Set

Decimal Code	Hexa-decimal Code	Octal Code	Graphic or Mnemonic	Name or Meaning
000	00	000	NUL	Null
001	01	001	SOH	Start of heading
002	02	002	STX	Start of text
003	03	003	ETX	End of text
004	04	004	EOT	End of transmission
005	05	005	ENQ	Enquiry
006	06	006	ACK	Acknowledge
007	07	007	BEL	Bell
008	08	010	BS	Backspace
009	09	011	HT	Horizontal tabulation
010	0A	012	LF	Line feed
011	0B	013	VT	Vertical tabulation
012	0C	014	FF	Form feed
013	0D	015	CR	Carriage return
014	0E	016	SO	Shift out
015	0F	017	SI	Shift in
016	10	020	DLE	Data link escape
017	11	021	DC1	Device control 1
018	12	022	DC2	Device control 2
019	13	023	DC3	Device control 3
020	14	024	DC4	Device control 4
021	15	025	NAK	Negative acknowledge
022	16	026	SYN	Synchronous idle
023	17	027	ETB	End of transmission block
024	18	030	CAN	Cancel
025	19	031	EM	End of medium
026	1A	032	SUB	Substitute
027	1B	033	ESC	Escape
028	1C	034	FS	File separator
029	1D	035	GS	Group separator
030	1E	036	RS	Record separator
031	1F	037	US	Unit separator
032	20	040	SP	Space
033	21	041	!	Exclamation point
034	22	042	"	Quotation marks
035	23	043	#	Number sign
036	24	044	\$	Dollar sign
037	25	045	%	Percent sign
038	26	046	&	Ampersand
039	27	047	'	Apostrophe
040	28	050	(Opening parenthesis
041	29	051)	Closing parenthesis
042	2A	052	*	Asterisk
043	2B	053	+	Plus

(Continued)

Table C-1. ASCII Character Set (Continued)

Decimal Code	Hexa-decimal Code	Octal Code	Graphic or Mnemonic	Name or Meaning
044	2C	054	,	Comma
045	2D	055	—	Hyphen
046	2E	056	.	Period
047	2F	057	/	Slant
048	30	060	0	Zero
049	31	061	1	One
050	32	062	2	Two
051	33	063	3	Three
052	34	064	4	Four
053	35	065	5	Five
054	36	066	6	Six
055	37	067	7	Seven
056	38	070	8	Eight
057	39	071	9	Nine
058	3A	072	:	Colon
059	3B	073	;	Semicolon
060	3C	074	<	Less than
061	3D	075	=	Equals
062	3E	076	>	Greater than
063	3F	077	?	Question mark
064	40	100	@	Commercial at
065	41	101	A	Uppercase A
066	42	102	B	Uppercase B
067	43	103	C	Uppercase C
068	44	104	D	Uppercase D
069	45	105	E	Uppercase E
070	46	106	F	Uppercase F
071	47	107	G	Uppercase G
072	48	110	H	Uppercase H
073	49	111	I	Uppercase I
074	4A	112	J	Uppercase J
075	4B	113	K	Uppercase K
076	4C	114	L	Uppercase L
077	4D	115	M	Uppercase M
078	4E	116	N	Uppercase N
079	4F	117	O	Uppercase O
080	50	120	P	Uppercase P
081	51	121	Q	Uppercase Q
082	52	122	R	Uppercase R
083	53	123	S	Uppercase S
084	54	124	T	Uppercase T
085	55	125	U	Uppercase U
086	56	126	V	Uppercase V
087	57	127	W	Uppercase W

(Continued)

Table C-1. ASCII Character Set (Continued)

Decimal Code	Hexa-decimal Code	Octal Code	Graphic or Mnemonic	Name or Meaning
088	58	130	X	Uppercase X
089	59	131	Y	Uppercase Y
090	5A	132	Z	Uppercase Z
091	5B	133	[Opening bracket
092	5C	134	\	Reverse slant
093	5D	135]	Closing bracket
094	5E	136	^	Circumflex
095	5F	137	_	Underline
096	60	140	`	Grave accent
097	61	141	a	Lowercase a
098	62	142	b	Lowercase b
099	63	143	c	Lowercase c
100	64	144	d	Lowercase d
101	65	145	e	Lowercase e
102	66	146	f	Lowercase f
103	67	147	g	Lowercase g
104	68	150	h	Lowercase h
105	69	151	i	Lowercase i
106	6A	152	j	Lowercase j
107	6B	153	k	Lowercase k
108	6C	154	l	Lowercase l
109	6D	155	m	Lowercase m
110	6E	156	n	Lowercase n
111	6F	157	o	Lowercase o
112	70	160	p	Lowercase p
113	71	161	q	Lowercase q
114	72	162	r	Lowercase r
115	73	163	s	Lowercase s
116	74	164	t	Lowercase t
117	75	165	u	Lowercase u
118	76	166	v	Lowercase v
119	77	167	w	Lowercase w
120	78	170	x	Lowercase x
121	79	171	y	Lowercase y
122	7A	172	z	Lowercase z
123	7B	173	{	Opening brace
124	7C	174		Vertical line
125	7D	175	}	Closing brace
126	7E	176	~	Tilde
127	7F	177	DEL	Delete

EBCDIC Character Set

NOS/VE uses the ASCII character set. When you write an EBCDIC tape, NOS/VE translates the ASCII characters to EBCDIC characters. When you read an EBCDIC tape, NOS/VE translates the EBCDIC characters to ASCII characters.

This section lists the ASCII character set with its EBCDIC equivalents (refer to table C-2) and the EBCDIC character set with its ASCII equivalents (refer to table C-3).

EBCDIC Character Set

Table C-2. ASCII to EBCDIC Conversion for Tapes

ASCII Character	ASCII Hexadecimal Value	EBCDIC Character	EBCDIC Hexadecimal Value
NUL	00	NUL	00
SOH	01	SOH	01
STX	02	STX	02
ETX	03	ETX	03
EOT	04	EOT	37
ENQ	05	ENQ	2D
ACK	06	ACK	2E
BEL	07	BEL	2F
BS	08	BS	16
HT	09	HT	05
LF	0A	LF	25
VT	0B	VT	0B
FF	0C	FF	0C
CR	0D	CR	0D
SO	0E	SO	0E
SI	0F	SI	0F
DLE	10	DLE	10
DC1	11	DC1	11
DC2	12	DC2	12
DC3	13	TM	13
DC4	14	DC4	3C
NAK	15	NAK	3D
SYN	16	SYN	32
ETB	17	ETB	26
CAN	18	CAN	18
EM	19	EM	19
SUB	1A	SUB	3F
ESC	1B	ESC	27
FS	1C	FS	1C
RS	1E	RS	1E
GS	1D	GS	1D
US	1F	US	1F
space	20	space	40
!	21	!	4F
"	22	"	7F
#	23	#	7B
\$	24	\$	5B
%	25	%	6C
&	26	&	50
'	27	'	7D
(28	(4D
)	29)	5D
*	2A	*	5C
+	2B	+	4E

(Continued)

Table C-2. ASCII to EBCDIC Conversion for Tapes (Continued)

ASCII Character	ASCII Hexadecimal Value	EBCDIC Character	EBCDIC Hexadecimal Value
,	2C	,	6B
-	2D	-	60
.	2E	.	4B
/	2F	/	61
0	30	0	F0
1	31	1	F1
2	32	2	F2
3	33	3	F3
4	34	4	F4
5	35	5	F5
6	36	6	F6
7	37	7	F7
8	38	8	F8
9	39	9	F9
:	3A	:	7A
;	3B	;	5E
<	3C	<	4C
=	3D	=	7E
>	3E	>	6E
?	3F	?	6F
@	40	@	7C
A	41	A	C1
B	42	B	C2
C	43	C	C3
D	44	D	C4
E	45	E	C5
F	46	F	C6
G	47	G	C7
H	48	H	C8
I	49	I	C9
J	4A	J	D1
K	4B	K	D2
L	4C	L	D3
M	4D	M	D4
N	4E	N	D5
O	4F	O	D6
P	50	P	D7
Q	51	Q	D8
R	52	R	D9
S	53	S	E2
T	54	T	E3
U	55	U	E4
V	56	V	E5
W	57	W	E6

(Continued)

Table C-2. ASCII to EBCDIC Conversion for Tapes (Continued)

ASCII Character	ASCII Hexadecimal Value	EBCDIC Character	EBCDIC Hexadecimal Value
X	58	X	E7
Y	59	Y	E8
Z	5A	Z	E9
[5B	[4A
\	5C	\	E0
]	5D]	5A
^	5E	^	5F
_	5F	_	6D
`	60	`	79
a	61	a	81
b	62	b	82
c	63	c	83
d	64	d	84
e	65	e	85
f	66	f	86
g	67	g	87
h	68	h	88
i	69	i	89
j	6A	j	91
k	6B	k	92
l	6C	l	93
m	6D	m	94
n	6E	n	95
o	6F	o	96
p	70	p	97
q	71	q	98
r	72	r	99
s	73	s	A2
t	74	t	A3
u	75	u	A4
v	76	v	A5
w	77	w	A6
x	78	x	A7
y	79	y	A8
z	7A	z	A9
{	7B	{	C0
	7C		6A
}	7D	}	D0
~	7E	~	A1
DEL	7F	DEL	07
undefined	80	undefined	20
undefined	81	undefined	21
undefined	82	undefined	22
undefined	83	undefined	23

(Continued)

Table C-2. ASCII to EBCDIC Conversion for Tapes (Continued)

ASCII Character	ASCII Hexadecimal Value	EBCDIC Character	EBCDIC Hexadecimal Value
undefined	84	undefined	24
undefined	85	undefined	15
undefined	86	undefined	06
undefined	87	undefined	17
undefined	88	undefined	28
undefined	89	undefined	29
undefined	8A	undefined	2A
undefined	8B	undefined	2B
undefined	8C	undefined	2C
undefined	8D	undefined	09
undefined	8E	undefined	0A
undefined	8F	undefined	1B
undefined	90	undefined	30
undefined	91	undefined	31
undefined	92	undefined	1A
undefined	93	undefined	33
undefined	94	undefined	34
undefined	95	undefined	35
undefined	96	undefined	36
undefined	97	undefined	08
undefined	98	undefined	38
undefined	99	undefined	39
undefined	9A	undefined	3A
undefined	9B	undefined	3B
undefined	9C	undefined	04
undefined	9D	undefined	14
undefined	9E	undefined	3E
undefined	9F	undefined	E1
undefined	A0	undefined	41
undefined	A1	undefined	42
undefined	A2	undefined	43
undefined	A3	undefined	44
undefined	A4	undefined	45
undefined	A5	undefined	46
undefined	A6	undefined	47
undefined	A7	undefined	48
undefined	A8	undefined	49
undefined	A9	undefined	51
undefined	AA	undefined	52
undefined	AB	undefined	53
undefined	AC	undefined	54
undefined	AD	undefined	55
undefined	AE	undefined	56
undefined	AF	undefined	57

(Continued)

Table C-2. ASCII to EBCDIC Conversion for Tapes (Continued)

ASCII Character	ASCII Hexadecimal Value	EBCDIC Character	EBCDIC Hexadecimal Value
undefined	B0	undefined	58
undefined	B1	undefined	59
undefined	B2	undefined	62
undefined	B3	undefined	63
undefined	B4	undefined	64
undefined	B5	undefined	65
undefined	B6	undefined	66
undefined	B7	undefined	67
undefined	B8	undefined	68
undefined	B9	undefined	69
undefined	BA	undefined	70
undefined	BB	undefined	71
undefined	BC	undefined	72
undefined	BD	undefined	73
undefined	BE	undefined	74
undefined	BF	undefined	75
undefined	C0	undefined	76
undefined	C1	undefined	77
undefined	C2	undefined	78
undefined	C3	undefined	80
undefined	C4	undefined	8A
undefined	C5	undefined	8B
undefined	C6	undefined	8C
undefined	C7	undefined	8C
undefined	C8	undefined	8E
undefined	C9	undefined	8F
undefined	CA	undefined	90
undefined	CB	undefined	9A
undefined	CC	undefined	9B
undefined	CD	undefined	9C
undefined	CE	undefined	9D
undefined	CF	undefined	9E
undefined	D0	undefined	9F
undefined	D1	undefined	A0
undefined	D2	undefined	AA
undefined	D3	undefined	AB
undefined	D4	undefined	AC
undefined	D5	undefined	AD
undefined	D6	undefined	AE
undefined	D7	undefined	AF
undefined	D8	undefined	B0
undefined	D9	undefined	B1
undefined	DA	undefined	B2
undefined	DB	undefined	B3

(Continued)

Table C-2. ASCII to EBCDIC Conversion for Tapes (Continued)

ASCII Character	ASCII Hexadecimal Value	EBCDIC Character	EBCDIC Hexadecimal Value
undefined	DC	undefined	B4
undefined	DD	undefined	B5
undefined	DE	undefined	B6
undefined	DF	undefined	B7
undefined	E0	undefined	B8
undefined	E1	undefined	B9
undefined	E2	undefined	BA
undefined	E3	undefined	BB
undefined	E4	undefined	BC
undefined	E5	undefined	BD
undefined	E6	undefined	BE
undefined	E7	undefined	BF
undefined	E8	undefined	CA
undefined	E9	undefined	CB
undefined	EA	undefined	CC
undefined	EB	undefined	CD
undefined	EC	undefined	CE
undefined	ED	undefined	CF
undefined	EE	undefined	DA
undefined	EF	undefined	DB
undefined	F0	undefined	DC
undefined	F1	undefined	DD
undefined	F2	undefined	DE
undefined	F3	undefined	DF
undefined	F4	undefined	EA
undefined	F5	undefined	EB
undefined	F6	undefined	EC
undefined	F7	undefined	ED
undefined	F8	undefined	EE
undefined	F9	undefined	EF
undefined	FA	undefined	FA
undefined	FB	undefined	FB
undefined	FC	undefined	FC
undefined	FD	undefined	FD
undefined	FE	undefined	FE
undefined	FF	undefined	FF

Table C-3. EBCDIC to ASCII Conversion for Tapes

EBCDIC Character	EBCDIC Hexadecimal Value	ASCII Character	ASCII Hexadecimal Value
NUL	00	NUL	00
SOH	01	SOH	01
STX	02	STX	02
ETX	03	ETX	03
undefined	04	undefined	9C
HT	05	HT	09
undefined	06	undefined	86
DEL	07	DEL	7F
undefined	08	undefined	97
undefined	09	undefined	8D
undefined	0A	undefined	8E
VT	0B	VT	0B
FF	0C	FF	0C
CR	0D	CR	0D
SO	0E	SO	0E
SI	0F	SI	0F
DLE	10	DLE	10
DC1	11	DC1	11
DC2	12	DC2	12
TM	13	DC3	13
undefined	14	undefined	9D
undefined	15	undefined	85
BS	16	BS	08
undefined	17	undefined	87
CAN	18	CAN	18
EM	19	EM	19
undefined	1A	undefined	92
undefined	1B	undefined	8F
FS	1C	FS	1C
GS	1D	GS	1D
RS	1E	RS	1E
US	1F	US	1F
undefined	20	undefined	80
undefined	21	undefined	81
undefined	22	undefined	82
undefined	23	undefined	83
undefined	24	undefined	84
LF	25	LF	0A
ETB	26	ETB	17
ESC	27	ESC	1B
undefined	28	undefined	88
undefined	29	undefined	89
undefined	2A	undefined	8A
undefined	2B	undefined	8B

(Continued)

Table C-3. EBCDIC to ASCII Conversion for Tapes (Continued)

EBCDIC Character	EBCDIC Hexadecimal Value	ASCII Character	ASCII Hexadecimal Value
undefined	2C	undefined	8C
ENQ	2D	ENQ	05
ACK	2E	ACK	06
BEL	2F	BEL	07
undefined	30	undefined	90
undefined	31	undefined	91
SYN	32	SYN	16
undefined	33	undefined	93
undefined	34	undefined	94
undefined	35	undefined	95
undefined	36	undefined	96
EOT	37	EOT	04
undefined	38	undefined	98
undefined	39	undefined	99
undefined	3A	undefined	9A
undefined	3B	undefined	9B
DC4	3C	DC4	14
NAK	3D	NAK	15
undefined	3F	undefined	9E
SUB	3F	SUB	1A
space	40	space	20
undefined	41	undefined	A0
undefined	42	undefined	A1
undefined	43	undefined	A2
undefined	44	undefined	A3
undefined	45	undefined	A4
undefined	46	undefined	A5
undefined	47	undefined	A6
undefined	48	undefined	A7
undefined	49	undefined	A8
[4A	[5B
.	4B	.	2E
<	4C	<	3C
(4D	(28
+	4E	+	2B
!	4F	!	21
&	50	&	26
undefined	51	undefined	A9
undefined	52	undefined	AA
undefined	53	undefined	AB
undefined	54	undefined	AC
undefined	55	undefined	AD
undefined	56	undefined	AE
undefined	57	undefined	AF

(Continued)

Table C-3. EBCDIC to ASCII Conversion for Tapes (Continued)

EBCDIC Character	EBCDIC Hexadecimal Value	ASCII Character	ASCII Hexadecimal Value
undefined	58	undefined	B0
undefined	59	undefined	B1
]	5A]	5D
\$	5B	\$	24
*	5C	*	2A
)	5D)	29
;	5E	;	3B
^	5F	^	5E
-	60	-	2D
/	61	/	2F
undefined	62	undefined	B2
undefined	63	undefined	B3
undefined	64	undefined	B4
undefined	65	undefined	B5
undefined	66	undefined	B6
undefined	67	undefined	B7
undefined	68	undefined	B8
undefined	69	undefined	B9
	6A		7C
,	6B	,	2C
%	6C	%	25
-	6D	-	5F
>	6E	>	3E
?	6F	?	3F
undefined	70	undefined	BA
undefined	71	undefined	BB
undefined	72	undefined	BC
undefined	73	undefined	BD
undefined	74	undefined	BE
undefined	75	undefined	BF
undefined	76	undefined	C0
undefined	77	undefined	C1
undefined	78	undefined	C2
~	79	~	60
:	7A	:	3A
#	7B	#	23
@	7C	@	40
'	7D	'	27
=	7E	=	3D
"	7F	"	22
undefined	80	undefined	C3
a	81	a	61
b	82	b	62
c	83	c	63

(Continued)

Table C-3. EBCDIC to ASCII Conversion for Tapes (Continued)

EBCDIC Character	EBCDIC Hexadecimal Value	ASCII Character	ASCII Hexadecimal Value
d	84	d	64
e	85	e	65
f	86	f	66
g	87	g	67
h	88	h	68
i	89	i	69
undefined	8A	undefined	C4
undefined	8B	undefined	C5
undefined	8C	undefined	C6
undefined	8D	undefined	C7
undefined	8E	undefined	C8
undefined	8F	undefined	C9
undefined	90	undefined	CA
j	91	j	6A
k	92	k	6B
l	93	l	6C
m	94	m	6D
n	95	n	6E
o	96	o	6F
p	97	p	70
q	98	q	71
r	99	r	72
undefined	9A	undefined	CB
undefined	9B	undefined	CC
undefined	9C	undefined	CD
undefined	9D	undefined	CE
undefined	9E	undefined	CF
undefined	9F	undefined	D0
undefined	A0	undefined	D1
~	A1	~	7E
s	A2	s	73
t	A3	t	74
u	A4	u	75
v	A5	v	76
w	A6	w	77
x	A7	x	78
y	A8	y	79
z	A9	z	7A
undefined	AA	undefined	D2
undefined	AB	undefined	D3
undefined	AC	undefined	D4
undefined	AD	undefined	D5
undefined	AE	undefined	D6
undefined	AF	undefined	D7

(Continued)

Table C-3. EBCDIC to ASCII Conversion for Tapes (Continued)

EBCDIC Character	EBCDIC Hexadecimal Value	ASCII Character	ASCII Hexadecimal Value
undefined	B0	undefined	D8
undefined	B1	undefined	D9
undefined	B2	undefined	DA
undefined	B3	undefined	DB
undefined	B4	undefined	DC
undefined	B5	undefined	DD
undefined	B6	undefined	DE
undefined	B7	undefined	DF
undefined	B8	undefined	E0
undefined	B9	undefined	E1
undefined	BA	undefined	E2
undefined	BB	undefined	E3
undefined	BC	undefined	E4
undefined	BD	undefined	E5
undefined	BE	undefined	E6
undefined	BF	undefined	E7
{	C0	{	7B
A	C1	A	41
B	C2	B	42
C	C3	C	43
D	C4	D	44
E	C5	E	45
F	C6	F	46
G	C7	G	47
H	C8	H	48
I	C9	I	49
undefined	CA	undefined	E8
undefined	CB	undefined	E9
undefined	CC	undefined	EA
undefined	CD	undefined	EB
undefined	CE	undefined	EC
undefined	CF	undefined	ED
}	D0	}	7D
J	D1	J	4A
K	D2	K	4B
L	D3	L	4C
M	D4	M	4D
N	D5	N	4E
O	D6	O	4F
P	D7	P	50
Q	D8	Q	51
R	D9	R	52
undefined	DA	undefined	EE
undefined	DB	undefined	EF

(Continued)

Table C-3. EBCDIC to ASCII Conversion for Tapes (Continued)

EBCDIC Character	EBCDIC Hexadecimal Value	ASCII Character	ASCII Hexadecimal Value
undefined	DC	undefined	F0
undefined	DD	undefined	F1
undefined	DE	undefined	F2
undefined	DF	undefined	F3
\	E0	\	5C
undefined	E1	undefined	9F
S	E2	S	53
T	E3	T	54
U	E4	U	55
V	E5	V	56
W	E6	W	57
X	E7	X	58
Y	E8	Y	59
Z	E9	Z	5A
undefined	EA	undefined	F4
undefined	EB	undefined	F5
undefined	EC	undefined	F6
undefined	ED	undefined	F7
undefined	EE	undefined	F8
undefined	EF	undefined	F9
0	F0	0	30
1	F1	1	31
2	F2	2	32
3	F3	3	33
4	F4	4	34
5	F5	5	35
6	F6	6	36
7	F7	7	37
8	F8	8	38
9	F9	9	39
undefined	FA	undefined	FA
undefined	FB	undefined	FB
undefined	FC	undefined	FC
undefined	FD	undefined	FD
undefined	FE	undefined	FE
undefined	FF	undefined	FF

ANSI Tape Label Formats

D

Required Labels	D-4
VOL1 - Volume Header Label	D-5
HDR1 - First File Header Label	D-8
HDR2 - Second File Header Label	D-15
EOF1 - First End-of-File Label	D-19
EOF2 - Second End of File Label	D-21
EOV1 - First End-of-Volume Label	D-22
EOV2 - Second End of Volume Label	D-24
Optional Labels	D-25
HDR3 through HDR9 - Additional File Header Labels ¹	D-25
EOF3 through EOF9 - Additional End-of-File Labels ¹	D-25
EOV3 through EOV9 - Additional End-of-Volume Labels ²	D-26
User Labels	D-26

ANSI labels perform two functions. They provide information that uniquely identifies a file and the reel on which it resides, and they mark the beginning and ending of a file and the beginning and end of a reel.

ANSI labels are designed to conform to the American National Standard Magnetic Tape Labels for Information Interchange X3.27-1978 and the December 1983 revision. All labels are 80 characters long and are recorded at the same density as the data on the tape. The first three characters of an ANSI label identify the label type. The fourth character indicates a number within a label type.

Labels padded to longer than 80 characters and which otherwise meet the ANSI X3.27-1969 standard can be read (but not written) by NOS/VE.

Table D-1 is a summary of each label type, name, function, and whether or not it is required.

Table D-1. Label Characteristics

Type	Number	Name	Used As	Required/ Optional
VOL	1	Volume header label	Beginning-of-volume	Required
UVL	1-9	User volume label	Beginning-of-volume	Optional (1)
HDR	1	File header label	Beginning-of-information	Required
HDR	2	File header label	Beginning-of-information	Required (2)
HDR	3-9	File header label	Beginning-of-information	Optional (1)
UHL	1-9	User header label	Beginning-of-information	Optional (1)
EOF	1	End-of-file label	End-of-information	Required

1. Ignored when reading labels, omitted when writing labels.

2. Allowed to be absent when reading labels, included when writing labels.

(Continued)

Table D-1. Label Characteristics (Continued)

Type	Number	Name	Used As	Required/ Optional
EOF	2	End-of-file label	End-of- information	Required (1)
EOF	3-9	End-of-file label	End-of- information	Optional (2)
UTL	1-9	User trailer label	End-of- information	Optional (2)
EOV	1	End-of-volume label	End-of-volume	Required (1)(3)
EOV	2	End-of-volume label	End-of-volume	Required (1)(3)
EOV	3-9	End-of-volume label	End-of-volume	Optional (1)

1. Allowed to be absent when reading labels, included when writing labels.

2. Ignored when reading labels, omitted when writing labels.

3. End of volume labels are required at the end of all but the last volume of a multivolume set.

Tape Label Formats

The VOL1, HDR1, and EOF1 labels are required on all ANSI-labeled tapes. In addition, an EOV1 label is required if the physical end-of-tape reflector is encountered before an EOF1 label is written or if a multifile set is continued on another volume. In the descriptions of the contents of these labels, n is any numeric digit and a is any uppercase letter, digit, or any of the following special characters.

Space ! " \$ % & ' () * + #
@ , - . / : ; < = > ? _

Some fields are optional. An optional field which does not contain the designated information must contain blanks. Fields which are not described as optional are required and written as specified. n-type fields are right-justified and zero-filled, and a-type fields are left-justified and blank-filled.

VOL1 - Volume Header Label

The volume header label must be the first label on a labelled tape. All reels begin with a VOL1 label. If two or more reels belong to a volume set, the file section field in the HDR1 label gives the actual reel number.

VOL		1	volume serial number	
va	reserved			
reserved				
reserved			owner identification	
owner identification (oid)				
oid	reserved			
reserved				
reserved				lsv

Table D-2 explains the above volume header label illustration. The length field is expressed in characters.

Table D-2. VOL1 - Volume Header Label (1)

Character Position	Field Name	Length	Contents
1-3	Label identifier (2)	3	Must be VOL.
4	Label number (2)	1	Must be 1.
5-10	Volume serial number (2)	6	Volume identification assigned by owner to identify this physical reel of tape.
11	Accessibility (3)	1	An a character which indicates the restrictions, if any, on who may have access to the information on the tape. A blank means unlimited access. Any other character means special handling, in the manner agreed between the interchange parties. The default is blank (unlimited access).

(1) None of the fields are checked on overwrite.

(2) Checked on read.

(3) Not checked on read.

(Continued)

Table D-2. VOL1 - Volume Header Label (1) (Continued)

Character Position	Field Name	Length	Contents
12-31	Reserved for future standardization (2)	20	Must be blanks.
32-37	Reserved for future standardization (2)	6	Must be blanks.
38-51	Owner identification (old) (2)	14	Any a characters identifying the owner of the physical volume.
52-79	Reserved for future standardization (2)	28	Must be blanks.
80	Label standard version (lsv) (2)	1	Identifies the version of the ANSI standard under which the labels were created. 1. ANSI X3.27 - 1969 2. Unused 3. ANSI X3.27 - 1978 4. ANSI X3.27 - 1983 The default is 4.

(1) None of the fields are checked on overwrite.

(2) Not checked on read.

HDR1 - First File Header Label

The first file header label must appear before each file. When a file is continued on more than one volume, the file header label is repeated after the volume header label on each new volume for that file. If two or more files are grouped in a multifile set, each HDR1 label indicates the relative position of its associated file within the set.

HDR	1	file identifier (fi)		
file identifier (fi)				
fi	file set identifier			file section number (secno)
secno	file sequence number	generation number		gvn
gvn	creation date		expiration date	
expiration date		fa	block count	
system code				
system code			reserved	

Table D-3 explains the above first file header label illustration. The length field is expressed in characters.

Table D-3. HDR1 - First File Header Label (1)

Character Position	Field Name	Length	Contents
1-3	Label identifier (2)	3	Must be HDR.
4	Label number (2)	1	Must be 1.
5-21	File identifier (fi) (3)	17	Up to 17 a characters used as the file identification (fid) parameter on the CHANGE_TAPE_LABEL_ATTRIBUTES command. The default is the leftmost 17 characters of the file path name.
22-27	Set (identifier file) (4)	6	Up to six characters used as the FILE_SET_IDENTIFIER parameter on the CHANGE_TAPE_LABEL_ATTRIBUTES command. This value is the same for all files of a multfile set. The default is the value of the volume serial number in the VOL1 label of the first tape volume of the file set.

(1) None of the header label fields are checked on overwrite.

(2) Checked on read.

(3) Checked on read if specified.

(4) Not checked on read.

(Continued)

Table D-3. HDR1 - First File Header Label (1) (Continued)

Character Position	Field Name	Length	Contents
28-31	File section number (secno) (2)	4	Four n characters identifying the file section number. The file section number of the first HDR1 label of a file is 0001. If the file extends to more than one volume, this number is incremented by one for each subsequent volume. The default is 0001.
32-35	File sequence number (2)	4	Four n characters used to specify the position of a file within a file set. This value is 0001 for the first file, 0002 for the second, and so on. In all the labels for a particular file, this field contains the same number. The default is the current position on the file set.

(1) None of the header label fields are checked on overwrite.

(2) Checked on read if specified.

(Continued)

Table D-3. HDR1 - First File Header Label (1) (Continued)

Character Position	Field Name	Length	Contents
36-39	Generation number (optional) (2)	4	Four n characters specifying the generation number of a file. This value is specified by the GENERATION_NUMBER parameter of the CHANGE_TAPE_LABEL_ATTRIBUTES command. This value is 0001 for the first generation of a file, 0002 for the second, and so on. The default is 0001.
40-41	Generation version number (gvn) (3)	2	Two n characters used to distinguish successive iterations of the same generation. The generation version number of the first attempt to create a file is 00. This value is specified by the GENERATION_VERSION_NUMBER parameter of the CHANGE_TAPE_LABEL_ATTRIBUTES command.

(1) None of the header label fields are checked on overwrite.

(2) Checked on read if specified.

(3) Checked on read.

(Continued)

Table D-3. HDR1 - First File Header Label (1) (Continued)

Character Position	Field Name	Length	Contents
42-47	Creation date (2)	6	Date the file was created; it is recorded as a space followed by two n characters for the year followed by three n characters for the day within the year. This value is specified by the CREATION_DATE parameter of the CHANGE_TAPE_LABEL_ATTRIBUTES command. The default is the current date.

(1) None of the header label fields are checked on overwrite.

(2) Checked on read. The creation date is meaningful only on read operations; on write operations the current date is used.

(Continued)

Table D-3. HDR1 - First File Header Label (1) (Continued)

Character Position	Field Name	Length	Contents
48-53	Expiration date (2)	6	<p>The file is considered expired when the EXPIRATION_DATE is earlier than or equal to today's date. When this condition is satisfied, the remainder of the volume may be overwritten. Thus, to be effective on multifile volumes, the expiration date of a file must be earlier than or the same as the expiration date of all preceding files on the volume. The expiration date is written in the same format as the creation date.</p> <p>This value is specified by the EXPIRATION_DATE parameter on the CHANGE_TAPE_LABELS_ATTRIBUTES command. The default is 000000.</p>

(1) None of the header label fields are checked on overwrite.

(2) Not checked on read.

(Continued)

Table D-3. HDR1 - First File Header Label (1) (Continued)

Character Position	Field Name	Length	Contents
54	File accessibility code (fa)(2)	1	An a character which indicates the restrictions, if any, on who may have access to the information in this file. A blank means unlimited access. This value is specified by the FILE_ACCESSIBILITY_CODE parameter on the CHANGE_TAPE_LABEL_ATTRIBUTES command. The default is blank (unlimited access).
55-60	Block count (2)	6	Must be zeros.
61-73	System code (2)	13	Thirteen characters identifying the operating system that recorded this file. The default is 'NOS/VE V1.0AA'.
74-80	Reserved for future standardization (2)	7	Must be spaces.

(1) None of the header label fields are checked on overwrite.

(2) Not checked on read.

HDR2 - Second File Header Label

The HDR2 label follows the HDR1 label. It contains information that identifies the format of the data in the file which follows:

HDR		2	rf	block length		
record length			bt	rt	block length extension (blx)	
blx	record length extension (rlx)	pc	cs	cc	reserved	
reserved						
reserved						
bol	reserved					
reserved						
reserved						

Table D-4 explains the above second file header label illustration. The length field is expressed in characters.

Table D-4. HDR2 - Second File Header Label (1)

Character Position	Field Name	Length	Contents
1-3	Label identifier (2)	3	Must be HDR.
4	Label number (2)	1	Must be 2.
5	Record Format (rf) (2)	1	Indicates the record format of the file as follows: F - ANSI Fixed D - ANSI Variable S - ANSI Spanned
6-10	Block length (bl) (2)	5	Specifies the maximum length of the tape blocks on the file.
11-15	Record length (rl) (2)	5	Specifies the maximum length of each record on the file.
16-17	Block type (bt) (2)	2	Indicates the NOS/VE block type of the file as follows: US - User-specified SS - System-specified

(1) None of the second file header fields are checked on overwrite.

(2) Checked on read.

(Continued)

Table D-4. HDR2 - Second File Header Label (1) (Continued)

Character Position	Field Name	Length	Contents
18	Record type (rt) (2)	1	Indicates the NOS/VE record type of the file as follows: F - ANSI Fixed U - Undefined V - Variable D - ANSI Variable S - ANSI Spanned
19-21	Block length extension (blx) (2)	3	Specifies the most significant digits of the block length
22-24	Record length extension (rlx) (2)	3	Specifies the most significant digits of the record length.
25	Padding character (pc) (2)	1	Specifies the NOS/VE character which is used to pad fixed length records.
26	Character Set (cs) (2)	1	Indicates the character set in which the file is recorded as follows: A - ASCII E - EBCDIC

(1) None of the second file header fields are checked on overwrite.

(2) Checked on read.

(Continued)

Table D-4. HDR2 - Second File Header Label (1) (Continued)

Character Position	Field Name	Length	Contents
27	Character conversion (cc) (2)	1	Indicates that character set conversion is to be performed between the data on the tape and NOS/VE. T - True (perform conversion) F - False (do not perform conversion)
51-52	Buffer offset length (bol) (2)	2	Specifies the number of characters at the beginning of each block that are not part of the data on the file.

(1) None of the second header label fields are checked on overwrite.

(2) Checked on read.

EOF1 - First End-of-File Label

The end-of-file label is the last block of every file. It is the system end-of-information for the file. A single tape mark precedes EOF1. A double tape mark written after the EOF1 label marks the end of a multifile set.

EOF		1	file identifier (fi)		
file identifier (fi)					
fi	set identification			file section number (secno)	
secno	file sequence number		generation number		gvn
gvn	creation date		expiration date		
expiration date		fa	block count		
system code					
system code			reserved		

Table D-5 explains the above first end of file label illustration. The length field is expressed in characters.

Table D-5. EOF1 - First End-of-File Label

Character Position	Field Name	Length	Contents
1-3	Label identifier (1)	3	Must be EOF.
4	Label number (1)	1	Must be 1.
5-54	Same as corresponding fields in HDR1 (optional) (2)	50	Same as the corresponding fields in HDR1.
55-60	Block count (3)	6	Six n characters specifying the number of data blocks between this label and the preceding HDR label group. This total does not include labels or tape marks.
61-80	Same as corresponding fields in HDR1 (optional) (2)	20	Same as corresponding fields in HDR1.

(1) Checked on read.

(2) Checked on read disposition same as corresponding fields in HDR1.

(3) Not checked on read.

EOF2 - Second End of File Label

The EOF2 label follows the EOF1 label. It contains information that identifies the format of the data on the file.

EOF		2	rf	block length		
record length			bt	rt	block length extension (blx)	
blx	record length extension (rlx)	pc	cs	cc	reserved	
reserved						
reserved						
bol	reserved					
reserved						
reserved						

Table D-6 explains the above second end-of-file label illustration. The length field is expressed in characters.

Table D-6. EOF2 - Second End-of-File Label

Character Position	Field Name	Length	Contents
1-3	Label identifier (1)	3	Must be EOF
4	Label number (1)	1	Must be 2
5-80	Same as corresponding fields in HDR2. (1)	76	Same as corresponding fields in HDR2.

(1) Not checked on read.

EOV1 - First End-of-Volume Label

The end-of-volume label is present only if the physical end-of-tape reflector is encountered before closing the file when writing or if the program writing this file closed the volume and continued writing the file on the next volume. EOV1 is preceded by a single tape mark and followed by a double tape mark.

EOV	1	file identifier (fi)		
file identifier (fi)				
fi	set identification			file section number (secno)
secno	file sequence number	generation number		gvn
gvn	creation date		expiration date	
expiration date		fa	block count	
system code				
system code			reserved	

Table D-7 explains the above first end of volume illustration. The length field is expressed in characters.

Table D-7. EOVI - First End-of-Volume Label

Character Position	Field Name	Length	Contents
1-3	Label identifier (1)	3	Must be EOVI.
4	Label number (1)	1	Must be 1.
5-54	Same as corresponding fields in HDR1 (optional) (2)	50	Same as the corresponding fields in HDR1.
55-60	Block count (3)	6	Six n characters specifying the number of data blocks between this label and the preceding HDR label group. This total does not include labels or tape marks.
61-80	Same as corresponding fields in HDR1 (optional) (2)	20	Same as corresponding fields in HDR1.

(1) Checked on read.

(2) Checked on read disposition same as corresponding fields in HDR1.

(3) Not checked on read.

EOV2 - Second End of Volume Label

The EOV2 label follows the EOV1 label. It contains information that identifies the format of the data on the file.

EOV		2	rf	block length		
record length			bt	rt	block length extension (blx)	
blx	record length extension (rlx)	pc	cs	cc	reserved	
reserved						
reserved						
bol	reserved					
reserved						
reserved						

Table D-8 explains the above second end-of-file label illustration. The length field is expressed in characters.

Table D-8. EOV2 - Second End-of-Volume Label

Character Position	Field Name	Length	Contents
1-3	Label identifier (1)	3	Must be EOV
4	Label number (1)	1	Must be 2
5-80	Same as corresponding fields in HDR2. (1)	76	Same as corresponding fields in HDR2.

(1) Not checked on read.

Optional Labels

Six types of optional labels are allowed. NOS/VE ignores these labels while reading an ANSI File. These labels are not written by NOS/VE. They are additional file header (HDR3-9), end-of-volume (EOV3-9), end-of-file (EOF3-9), user volume (UVLa), header (UHLa), and trailer (UTLa) labels.

HDR3 through HDR9 - Additional File Header Labels¹

HDR3 through HDR9 labels may immediately follow HDR2. Their format is:

Character Position	Field Name	Length in Characters	Contents
1-3	Label identifier	3	HDR
4	Label number	1	2-9
5-80		76	

Only the label identifier and the label number are checked on read.

EOF3 through EOF9 - Additional End-of-File Labels¹

EOF3 through EOF9 labels may immediately follow EOF2. Their format is:

Character Position	Field Name	Length in Characters	Contents
1-3	Label identifier	3	EOF
4	Label number	1	2-9
5-80		76	

Only the label identifier and the label number are checked on read.

1. Reserved for operating system use.

EOV3 through EOV9 - Additional End-of-Volume Labels²

EOV3 through EOV9 labels may immediately follow EOV2. Their format is:

Character Position	Field Name	Length in Characters	Contents
1-3	Label identifier	3	EOV
4	Label number	1	2-9
5-80		76	

User Labels

User labels may immediately follow their associated system labels. Thus, user volume labels (UVLa) may follow VOL1, user header labels (UHLa) may follow the last HDRn label, and user trailer labels (UTLa) may follow the last EOVn or EOFn label. Their format is:

Character Position	Field Name	Length in Characters	Contents
1-3	Label identifier	3	UVL, UHL, or UTL.
4	Label number	1	Must be 1-9 for UVL labels. For other labels, any a character.
5-80	User option	76	Any a characters.

2. Reserved for operating system use.

Format Effectors

E

Vertical Spacing Characters	E-1
Vertical Form Unit Loading	E-3
Reserved Format Channels	E-3
Vertical Forms Unit Load Image	E-4

This appendix describes vertical spacing characters and vertical form unit loading

Vertical Spacing Characters

The following characters cause vertical spacing actions when they are encountered in the first character position of a print line. In this appendix, these characters are called format effectors. If your file contains a format effector that specifies a capability not available on all printers (such as the format effector that specifies Load Vertical Forms Unit), you must specify an `EXTERNAL_CHARACTERISTIC` parameter on the `PRINT_FILE` command to ensure that the appropriate printer is used to print the file.

Table E-1. Vertical Spacing Characters

Format Effector	Meaning
1	Advance to top-of-form (page-eject) before printing ¹
2	Advance to bottom-of-form channel before printing
+	Advance 0 lines before printing (overprint) ¹
sp	Advance 1 line before printing (single space) ¹
0	Advance 2 lines before printing ¹
-	Advance 3 lines before printing ¹
8	Advance to channel 1 before printing
7	Advance to channel 2 before printing
6	Advance to channel 3 before printing
5	Advance to channel 4 before printing
4	Advance to channel 5 before printing
3	Advance to channel 6 before printing
9	Advance to channel 7 before printing
X	Advance to channel 8 before printing

1. Supported for all printers. Action taken for characters not supported by a printer is specified in the device definition.

(Continued)

Table E-1. Vertical Spacing Characters (Continued)

Format Effector	Meaning
Y	Advance to channel 9 before printing
Z	Advance to channel 10 before printing
W	Advance to channel 11 before printing
U	Advance to channel 12 before printing
A	Advance to top-of-form (page eject) after printing ¹
B	Advance to bottom-of-form channel before printing
/	Advance 0 lines after printing (overprint) ¹
H	Advance to channel 1 after printing
G	Advance to channel 2 after printing
F	Advance to channel 3 after printing
E	Advance to channel 4 after printing
D	Advance to channel 5 after printing
C	Advance to channel 6 after printing
I	Advance to channel 7 after printing
J	Advance to channel 8 after printing
K	Advance to channel 9 after printing
L	Advance to channel 10 after printing
M	Advance to channel 11 after printing
N	Advance to channel 12 after printing
Q	Deselect auto page-eject
R	Select auto page-eject
S	Select 6 lines per inch
T	Select 8 lines per inch
V	Load Vertical Forms Unit
<US>	Single space
others	Undefined format effector action

Vertical Form Unit Loading

Printers which support format channels contain some sort of vertical format unit (VFU). If a printer has a VFU, it may be used to support different forms sizes as well as the selection of vertical print density and auto page-eject control.

Reserved Format Channels

For printers which support format channels, three channels are reserved for specific carriage control operations. Channel 1 is always used to position the paper to its top-of-form. Two additional channels of the site's choosing are also reserved: one for positioning the paper to the logical bottom-of-form, the other for skipping over the perforations between forms (auto page-eject). Each of these channels may occur only once in the Vertical Forms Unit load image.

By default, both the bottom-of-form and auto page-eject channels are placed at two lines above the physical end of the form. If the operator or user changes forms size or print density, the bottom-of-form and auto page-eject channels are moved to accommodate the change in the number of lines on the form.

Vertical Forms Unit Load Image

Figures E-1 and E-2 show the standard VFU load image used if the site or user does not specify a VFU load procedure for a printing device or a print file.

Line	Channels											
	1	2	3	4	5	6	7	8	9	10	11	12
1	X	X	X	X	X	X	X		X	X	X	
2												
3		X										
4			X						X			
5		X		X								
6					X							
7		X	X			X			X			
8							X					
9		X		X								
10		X	X						X			
11		X			X					X		
12											X	
13		X	X	X		X			X			
14												
15		X					X					
16		X	X		X				X			
17		X		X								
18		X	X			X			X			
19												
20												
21		X		X	X					X		
22		X	X				X		X			
23		X										
24												
25		X	X	X		X			X			
26					X							
27		X										
28		X	X						X			
29		X		X			X					
30												
31		X	X		X	X			X	X		
32												
33		X		X								
34		X	X						X			
35		X										
36					X	X						
37		X	X	X		X			X			
38												
39		X										
40			X						X			
41		X		X	X					X		
42												
43		X	X			X	X		X			
44												
45		X		X								
46			X		X				X			
47		X										
48												

Figure E-1. Vertical Forms Unit Load Image
(Continued)

(Continued)

Line	Channels								
49	X	X	X	X	X	1	X	1	1 - Set for 8.5 inch form at 6 lines per inch
50									
51	X			X				X	
52		X						X	
53	X		X						
54									
55	X	X		X				X	
56				X					
57	X		X		X				
58		X						X	
59	X								
60									
61	X	X	X	X	X			X X	2 - Set for 11 inch form at 6 lines per inch
62									
63	X								
64		X			X	2	X	2	
65	X		X						
66				X		3		3	
67	X	X		X			X		
68									
69	X		X						
70		X				4	X	4	
71	X			X	X			X	
72									
73	X	X	X	X				X	
74									
75	X								
76		X		X				X	
77	X		X						
78					X				
79	X	X		X				X	
80									
81	X	X	X	X				X X	
82		X						X	
83	X	X							
84									
85	X	X	X	X	X			X	
86				X		5		5	5 - Set for 11 inch form at 8 lines per inch
87	X								
88		X					X		
89	X		X						
90									
91	X	X	X	X			X	X	
92					X				
93	X		X						
94		X				6	X	6	
95	X								
96			X						

Figure E-1. Vertical Forms Unit Load Image

SCL Language Syntax

F

Metalanguage Symbols	F-1
SCL Language Syntax	F-3
Calls to Commands	F-5

This appendix contains reference information about the syntax of language elements supplied as input to the SCL interpreter. In addition to presenting the basic syntax, this appendix describes the form of calls to commands.

Metalanguage Symbols

Table F-1 presents the conventions (metalanguage symbols) used in the syntax presentation. Unless underlined, any character from this table should be read as a metalanguage symbol.

Table F-1. SCL Metalanguage Conventions

Symbol	Description
::=	Is read as <i>is defined to be</i> .
	Is read as <i>or</i> ; elements separated by this symbol are mutually exclusive.
< >	Elements enclosed by these symbols constitute a single element in relation to the surrounding metalanguage symbols.
[]	Elements enclosed by these symbols are optional and constitute a single element in relations to the surrounding metalanguage symbols.
...	Elements followed by this symbol may be repeated. When this symbol follows a <i>greater than</i> symbol (>) or closing bracket (]), it applies to the metalanguage text between and including the matching <i>lesser than</i> symbol (<) or opening bracket ([).

(Continued)

Table F-1. SCL Metalanguage Conventions (Continued)

Symbol	Description
<any ASCII character except xxxxx>	Denotes any ASCII character except the character(s) specified by xxxxx.
<sp>	One or more spaces and/or comments.
<nl>	New line.
<bol>	Beginning-of-line.
<eol>	End-of-line.
<eop>	End-of-partition.
<eoi>	End-of-information.
<..>	Two or more dots optionally preceded and/or followed by <sp>.

SCL Language Syntax

The syntax of SCL is presented in metalanguage form as follows:

```

<SCL input> ::= <job>
              | <command procedure>
              | <included file>
              | <included line>

<job> ::= <login> [<statement list>] <logout>

<login> ::= <batch login>
           | <interactive login>

<batch login> ::= <bol> [<;|nl>]...
                <login command> <;|nl>

<logout> ::= LOGOUT | <end of text>

<end of text> ::= <eol> <eoi>
                | <eol> <eop>

<included file> ::= <bol> [<statements>] <end of text>

<included line> ::= <bol> [<statements>] <eol>

<statement list> ::= <statements> <;|nl>

<statements> ::= <statement> [<;|nl> <statement>]...

```

SCL Language Syntax

```
<statement> ::= <command>
              | <variable definitions>
              | <assignment statement>
              | <push>
              | <pop>
              | <condition statement>
              | <structured statement>
              | <if statement>
              | <control statement>
              | <empty>

<condition statement> ::= <when statement>
                        | <cancel>
                        | <continue>

<structured statement> ::= <block statement>
                          | <repetitive statement>

<repetitive statement> ::= <for statement>
                          | <loop statement>
                          | <repeat statement>
                          | <while statement>

<control statement> ::= <cycle>
                      | <exit>
                      | <exit_proc>

<;|nl> ::= <;>|<nl>

<;> ::= [<sp>] ; [<sp>]
```

Calls to Commands

The definitions shown in figure F-1 illustrate the form of calls to commands:

```

<command> ::= <command reference> [<,|sp> <command parameters>]

<command reference> ::=
    <command name>
| </> [<sp>] <command name>
| $SYSTEM <.> <command name>
| <file> <.> <command name>
| <catalog> <.> <command name>

<command parameters> ::=
    [<command parameter>] [<,|sp> [<command parameter>]]...

<command parameter> ::=
    [<parameter name> [<sp>] = [<sp>]] <parameter value>

<parameter value> ::= <expression>
    | <omitted parameter name>

<,|sp> ::= <, > | <sp>

<,> ::= [<sp>] , [<sp>]

```

Figure F-1. Calls to Commands

The following conditions apply to calls to commands:

- If you specify only <command name>, the command list is searched for the command. See the chapter 5, Command and SCL Procedure Execution for a discussion of SCL command lists.
- If you prefix the <command name> entry by a slant character (/), the entry at the front of the command list is bypassed in the search for the command. This facility is normally used within a command utility to call a command that does not belong to the utility but has the same name (usually abbreviated) as a utility subcommand.
- If you specify \$SYSTEM, the <command name> entry must designate one of the commands supplied as part of NOS/VE.
- If you specify a <file> entry, the file must comprise an object library containing the command. You may use a cycle reference, as part of the <file> specification, to select a particular version of the object library. See the NOS/VE Object Code Management manual for details about how an object library is searched for a command.
- If you specify a <catalog> entry, the <command name> entry must be the name of a file within the designated catalog that can be treated as a command.

For information about the syntax of procedures, see appendix G.

SCL Procedure Syntax

G

Basic Syntax	G-1
Expressions	G-4
Boolean Expression	G-4
File Expression	G-6
Integer Expression	G-8
Keyword Expression	G-10
Name Expression	G-10
Real Expression	G-11
String Expression	G-12
Status Expression	G-12

This appendix documents the syntax of SCL procedures. In addition to presenting the basic syntax, this appendix contains syntax descriptions of expressions.

The definition of a procedure consists of two main parts:

- A header in which the names for the command are defined.
- A specification for the command's parameters.

Basic Syntax

The basic procedure syntax appears as follows (see appendix F for an explanation of special symbols):

```
<proc header> ::= [<sp>] PROC <sp> <proc names>
                [[<sp>] <(> <param defs> <)>]

<proc names> ::= <proc name> [<,>|<sp> <proc name>]...

<proc name> ::= <name>

<param defs> ::=
    <param def> [<,>|<n1> <param def>]...

<param def> ::= <param names> [[<sp>]:<sp>] <value spec>]
                [[<sp>] = [<sp>] <default spec>]
                | <empty>

<param names> ::= <parameter name> [<,>|<sp> <parameter name>]...

<parameter name> ::= <name>
```


Basic Syntax

```
<value spec> ::=
    [<value list kind> <sp> OF <sp>] <param value kind>

<value list kind> ::= LIST [<sp> <value set count>
    [<,|sp> <value count>] [<,|sp> RANGE]]
    | [LIST <sp>] RANGE

<value set count> ::= <min value sets> [<..> <max value sets>]
<value count> ::= <min values> [<..> <max values>]
<min value sets> ::= <integer param expr>
<max value sets> ::= <integer param expr>
<min values> ::= <integer param expr>
<max values> ::= <integer param expr>

<param value kind> ::= <value kind> [<or keyword values>]
    | <var value kind> [<or keyword values>]
    | <keyword values>

<or keyword values> ::= <sp> OR <sp> <keyword values>

<keyword values> ::=
    KEY <sp> <keyword value> [<,|sp> <keyword value>]...

<keyword value> ::= <name>

<var value kind> ::= VAR <sp> OF <sp> <var kind>
    | ARRAY <sp> OF <sp> <var kind>
```

```

<var kind> ::=
    BOOLEAN
  | INTEGER
  | REAL
  | STATUS
  | STRING
  | ANY

```

```

<value kind> ::=
  | BOOLEAN
  | FILE
  | INTEGER [<sp> <min int value> <..> <max int value>]
  | NAME [<sp> <min name length> [<..> <max name length>]]
  | REAL
  | STATUS
  | STRING [<sp> <min str length> [<..> <max str length>]]
  | ANY
  | <application value name> [<sp> <application procedure name>]

```

```

<min name length> ::= <integer param expr>
<max name length> ::= <integer param expr>
<min str length> ::= <integer param expr>
<max str length> ::= <integer param expr>
<min int value> ::= <integer param expr>
<max int value> ::= <integer param expr>
<application procedure name> ::= <name>

```

```

<default spec> ::= $REQUIRED | $OPTIONAL | <default value>

```

```

<default value> ::= <value list>

```

```

<( > ::= ([<sp>]

```

```

<)> ::= [<sp>])

```

```

<..> ::= .. [.] ...

```

.....

Expressions

An expression is used to designate or compute a value of a particular type.

It is defined as follows:

```
<expression> ::= <boolean expression>
                | <file expression>
                | <integer expression>
                | <keyword expression>
                | <name expression>
                | <real expression>
                | <string expression>
                | <status expression>
```

Definitions for the preceding kinds of expressions are presented in the next sections.

Boolean Expression

This expression is defined as follows:

```
<boolean expression> ::=
    <boolean term 1> [<boolean sum | diff> <boolean term 1>]...

<boolean sum | diff> ::= <boolean sum> | <boolean difference>

<boolean sum> ::= <sp> OR <sp>

<boolean difference> ::= <sp> XOR <sp>

<boolean term 1> ::=
    <boolean term 2> [<boolean product> <boolean term 2>]...

<boolean product> ::= <sp> AND <sp>

<boolean term 2> ::= [<boolean complement>] <boolean operand>
```

```

<boolean complement> ::= NOT <sp>

<boolean operand> ::= <boolean variable>
                    | <boolean function>
                    | <boolean>
                    | <relational expression>
                    | <( > <boolean expression> <)>

<boolean> ::= <>true> | <>false>

<>true> ::= TRUE | YES | ON

<>false> ::= FALSE | NO | OFF

<relational expression> ::=
    <relational term> [<relation> <relational term>]

<relation> ::= <equal to>
              | <not equal to>
              | <greater than>
              | <greater than or equal to>
              | <less than>
              | <less than or equal to>

<equal to> ::= [<sp>] = [<sp>]

<not equal to> ::= [<sp>] ≤ ≥ [<sp>]

<greater than> ::= [<sp>] ≥ [<sp>]

<greater than or equal to> ::= [<sp>] ≥= [<sp>]

<less than> ::= [<sp>] ≤ [<sp>]

<less than or equal to> ::= [<sp>] ≤= [<sp>]

<relational term> ::= <expression>

```

File Expression

This expression is defined as follows:

```
<file expression> ::= <file reference>
                    | <file>
                    | <catalog>
```

```
<file | catalog> ::= <file> | <catalog>
```

```
<file reference> ::=
    <file> [<.> <file position>]
```

```
<file> ::= <path> [<.> <cycle reference>]
```

```
<catalog> ::= <path>
```

```
<path> ::= <absolute path>
          | <relative path>
```

```
<absolute path> ::= <family path>
                   | <system file>
                   | <user path>
```

```
<family path> ::=
    <:> <family name> [<.> <relative path>]
```

```
<system file> ::= <job file>
                  | <standard file>
```

```
<job file> ::= COMMAND | INPUT | OUTPUT
              | $JOB_LOG | $NULL
```

```
<standard file> ::= $ECHO | $ERRORS | $INPUT  
                  | $LIST | $OUTPUT | $RESPONSE
```

```
<user path> ::=  
    <.> <user name> [<.> <relative path>]
```

```
<relative path> ::=  
    <path element> [<.> <relative path>]
```

```
<path element> ::= <path element name>
```

```
<cycle reference> ::=  
    <cycle number>  
    | <generic cycle reference>
```

```
<cycle number> ::= <unsigned integer>
```

```
<generic cycle reference> ::= $HIGH | $LOW | $NEXT
```

```
<file position> ::= $ASIS | $BOI | $EOI
```

```
<:> ::= :
```

```
<.> ::= .
```

Integer Expression

An integer expression is a numeric expression with an integer result. If the result of the expression as given is of type real, the result is automatically converted to an integer.

This expression is defined as follows:

```

<integer expression> ::=
    <integer term 1> [<add | sub> <integer term 1>]...

<add | sub> ::= <add> | <subtract>

<add> ::= [<sp>] + [<sp>]

<subtract> ::= [<sp>] - [<sp>]

<integer term 1> ::=
    <integer term 2> [<multiply | divide> <integer term 2>]...

<multiply | divide> ::= <multiply> | <divide>

<multiply> ::= [<sp>] * [<sp>]

<divide> ::= [<sp>] / [<sp>]

<integer term 2> ::=
    <integer term 3> [<exponentiate> <integer term 3>]...

<exponentiate> ::= [<sp>] ** [<sp>]

```

<integer term 3> ::= [<plus | minus>] <integer operand>

<plus | minus> ::= <plus> | <minus>

<plus> ::= + [<sp>]

<minus> ::= - [<sp>]

<integer operand> ::= <integer variable>
 | <integer function>
 | <unsigned integer>
 | <(> <integer expression> . < >>

<integer> ::= [<sign>] <unsigned integer>

<unsigned integer> ::= <digit> [<hex digit>]... [<(> <radix> < >]

<sign> ::= <+|-> [<sp>]

<hex digit> ::= <digit> | A | B | C | D | E | F
 | a | b | c | d | e | f

<radix> ::= <unsigned decimal>

<unsigned decimal> ::= <digit>

Keyword Expression

This expression is defined as follows:

```
<keyword expression> ::= <keyword>
                        | <name function>
```

Name Expression

This expression is defined as follows:

```
<name expression> ::= <name>
                    | <name function>
<name> ::= <alphanumeric char> [<alphanumeric char>]...
```

```
<alphanumeric char> ::= <alphanumeric char> | <digit>
```

```
<alphanumeric char> ::= <letter>
                    | <international letter>
                    | <special alphanumeric char>
```

```
<letter> ::= <upper case letter>
            | <lower case letter>
```

```
<upper case letter> ::= A | B | C | D | E | F | G | H | I | J | K
                    | L | M | N | O | P | Q | R | S | T | U | V
                    | W | X | Y | Z
```

```
<lower case letter> ::= a | b | c | d | e | f | g | h | i | j | k
                    | l | m | n | o | p | q | r | s | t | u | v
                    | w | x | y | z
```

```
<international letter> ::= <upper case international letter>
                        | <lower case international letter>
```

```
<upper case international letter> ::= | @ | [ | \ | ] |
```

```
<lower case international letter> ::= | | { | _ | } |
```

```
<special alphanumeric character> ::= | # | $ | _
```

```
<digit> ::= | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
```

Real Expression

A real expression is a numeric expression with a real result. This expression is defined as follows:

```

<real expression> ::= <real variable>
                    | <real function>
                    | <unsigned real>

<real> ::= [<sign>] <unsigned real>

<unsigned real> ::= <mantissa> [<exponent>]

<mantissa> ::= <integer part> <.> <fraction part>

<integer part> ::= <unsigned decimal>

<fraction part> ::= <unsigned decimal>

<exponent> ::= <exponent char> [<+|->] <unsigned decimal>

<exponent char> ::= E | e | D | d

```

String Expression

This expression is defined as follows:

```
<string expression> ::=  
    <string operand> [<concatenate> <string operand>]...
```

```
<concatenate> ::= [<sp>] // [<sp>]
```

```
<string operand> ::= <string variable>  
                    | <string function>  
                    | <string>  
                    | <( > <string expression> < )>
```

```
<string> ::= <'> [<string char>]... <'>
```

```
<string char> ::= [<any ascii character except '> | '>
```

The concatenation operator (//) joins two strings.

Status Expression

This expression is defined as follows:

```
<status expression> ::= <status variable>  
                    | <status function>
```

Old Commands

H

Ordering Printed Manuals	H-1
Accessing Online Manuals	H-1

Old Commands

Table H-1 lists old commands from previous versions of NOS/VE and the preferred command or replacement command. Some of the preferred commands may have parameters that differ from an old command. Commands listed more than once were replaced by more than one command.

Table H-1. NOS/VE Old Commands

Old Command	Preferred/Replacement Command
CHANGE_TERM_CONN_ATTRIBUTES	CHANGE_CONNECTION_ATTRIBUTES
DISPLAY_170_REQUEST	DISPLAY_TAPE_LABEL_ATTRIBUTES
DISPLAY_7600_REQUEST	DISPLAY_TAPE_LABEL_ATTRIBUTES
DISPLAY_COMMAND_PARAMETERS	DISPLAY_COMMAND_INFORMATION
DISPLAY_IBM_REQUEST	DISPLAY_TAPE_LABEL_ATTRIBUTES
DISPLAY_PRINT_STATUS	DISPLAY_OUTPUT_STATUS
DISPLAY_TERM_CONN_ATTRIBUTES	DISPLAY_CONNECTION_ATTRIBUTES
DISPLAY_VAX_REQUEST	DISPLAY_TAPE_LABEL_ATTRIBUTES
EDIT_LIBRARY (SCU subcommand)	EDIT_DECK (SCU Subcommand)
SET_COMMAND_LIST	CREATE_COMMAND_LIST_ENTRY
SET_COMMAND_LIST	DELETE_COMMAND_LIST_ENTRY

(Continued)

Table H-1. NOS/VE Old Commands (Continued)

Old Command	Preferred/Replacement Command
SET_COMMAND_LIST	CHANGE_COMMAND_SEARCH_MODE
SET_COMMAND_MODE	CHANGE_INTERACTION_STYLE
SET_COMMAND_MODE	CHANGE_SCL_OPTIONS
SET_JOB_LIMIT	CHANGE_JOB_LIMIT
SET_LINK_ATTRIBUTES	CHANGE_LINK_ATTRIBUTES
SET_MESSAGE_MODE	CHANGE_MESSAGE_LEVEL
SET_TERMINAL_ATTRIBUTES	CHANGE_TERMINAL_ATTRIBUTES
TERMINATE_PRINT	TERMINATE_OUTPUT

Index

Index

A

Abbreviation

Command 2-12

Parameter 2-16

Abort, glossary definition A-1

Absolute path, glossary

definition A-1

ACCEPT_LINE 9-3, 10

Access control entries 4-22

Access mode 4-23

Changing 4-26

Creating and deleting 4-26

Multiple entries 4-26

Permit groups 4-22

Share mode 4-24

Access log 4-19

ACCESS_MODE file

attribute 4-40

\$ACCESS_MODE function 9-6

Account name 3-5

Active job logs 6-66

Displaying 6-69

ADMINISTER_

VALIDATION 3-21, 23; 6-11;

10-3

Alias, glossary definition A-1

Alphabetic character, glossary

definition A-1

Alphanumeric character, glossary

definition A-1

ANSI file 11-6

ANSI, glossary definition A-1

ANSI-labelled tapes 11-5; D-1

ANSI file 11-6

EOF1 label 11-13; D-19

EOF2 label 11-13; D-21

EOF3 through EOF9 D-25

EOV1 label D-22

EOV2 label D-24

Examples 11-28

File set 11-6

HDR1 label information 11-8;

D-8

HDR2 label information 11-9;

D-15

HDR3 through HDR9 D-25

Initializing volumes 11-16

Label information 11-6

Label processing 11-13

Label types D-2

Required D-4

Multifile volumes 11-12

NOS/VE HDR2

extensions 11-10

Reading 11-18

Tape file 11-6

Tape label attribute 11-19

Tape label formats D-1

Volume information 11-7

Writing 11-17

ANSI tape user labels D-26

Any

Data type 2-22; 7-36

Definition 7-36

ANY_FAULT condition 8-20

Application value

Data type 2-22; 7-36

Definition 7-36

Glossary definition A-2

Array

Definition 7-6

Lower bound 7-6

References 7-20

Upper bound 7-6

Variables 7-6

ARRAY OF

Clause 9-32

Restrictions 9-32

ASCII

Character set C-1

Glossary definition A-1

\$ASIS 4-38

Assignment statement

Definition 2-32

Examples 7-7

Glossary definition A-2

Processing syntax errors 9-16

Asynchronous tasks 6-33

Glossary definition A-2

ATTACH_FILE 4-31; 8-18; 9-10

ATTACH_JOB 3-19

Attaching files 4-30
 Explicitly 4-32
 More than once within a
 job 4-36
 Private readers 4-37
 Sharing while attached 4-33
ATTENTION_CHARACTER
 Connection attribute 13-25
 Terminal attribute 13-5
 Attributes, definition 1-3
AVERAGE_RECORD_LENGTH
 file attribute 4-41

B

BACKSPACE_CHARACTER
 terminal attribute 13-5
BACKUP_CATALOG sub
 BACPF 12-5, 10, 17
BACKUP_FILE sub
 BACPF 12-5, 12, 17
\$BACKUP_FILE sub
 RESPF 12-23, 27
BACKUP_PERMANENT_
FILES 12-1, 5
 Backup utility 12-1, 5
 Backing up
 Catalogs 12-10
 Files 12-12
 To disk 12-6
 To tape files 12-6
 Changing backup tape
 types 12-6
 Deleting catalogs 12-11
 Deleting file cycles 12-16
 Displaying backup tape
 types 12-6
 Excluding catalogs 12-10
 Excluding file cycles 12-16
 Excluding files 12-13
 Including file cycles 12-13
 By size 12-14
 List file 12-4
 Rules for file access 12-3
 Starting the utility 12-5
 Stopping the utility 12-5
 To \$NULL 12-9

To tape files
 Labelled 12-6
 Multifile tapes 12-7
 Multivolume 12-7
 Batch job 6-1
 Input 2-2
 Termination 8-20
 Batch mode, glossary
 definition A-2
BEGIN_LINE_CHARACTER
 terminal attribute 13-6
 Beginning of information,
 glossary definition A-2
 Bit, glossary definition A-2
 Block
 Count (ANSI file) D-14
 Definition 8-2
 Execution 8-3
 Glossary definition A-2
INCLUDE_FILE 9-14
INCLUDE_LINE 9-16
 Input 8-17
 Job 7-2, 13
 Length (ANSI file) 11-10;
 D-16
 Length extension (ANSI
 file) D-17
 Procedure 7-2; 9-23
 Statically linked 7-10; 8-3
 Type (ANSI file) 11-10; D-17
 Utility 7-2
UTILITY/UTILITYEND 9-53,
 58
WHEN/WHENEND 7-2
BLOCK/BLOCKEND
 Exiting 8-5
 Statement 8-5
 Syntax 8-5
BLOCK_TYPE file
 attribute 4-42; 11-10
BLOCK_TYPE tape label
 attribute 11-19
\$BOI 4-12
 Boolean
 Conventions 7-31
 Data type 2-22, 31; 7-31
 Definition 7-31
 Expressions 7-40
 Glossary definition A-2
 Operators 7-40

Boolean constant, glossary definition A-3
BREAK_KEY_ACTION connection attribute 13-26
 Buffer offset length (ANSI file) 11-10; D-18
BUFFER_OFFSET tape label attribute 11-19
 Byte-addressable file organization, glossary definition A-3
 Byte, glossary definition A-3

C

CANCEL_LINE_CHARACTER terminal attribute 13-6
CANCEL statement 8-25
 Canceling a condition 8-25
CARRIAGE_RETURN_DELAY terminal attribute 13-6
CARRIAGE_RETURN_SEQUENCE terminal attribute 13-6
\$CATALOG function 4-7; 9-6
 Catalog name, glossary definition A-3
 Catalog reference 7-34
 Element 7-34
 Maximum length 7-34
 Catalogs 4-1
 Access control entries 4-22
 Multiple entries 4-26
 Backing up 12-10
 Creation 4-13
 Deletion 4-15
 Displaying 4-20
 Entries 4-18
 Changing 4-20
 Displaying 4-19
 Glossary definition A-3
 Hierarchy 1-2; 4-3
 \$LOCAL 4-1
 Master (**\$USER**) 4-3
 Permanent 4-3
 Glossary definition A-18
 Permits 4-21
 Remote management 4-84
 Explicit 4-86
 Implicit 4-85
 Restoring from backup files 12-23, 25
 \$SYSTEM, glossary definition A-24
 Temporary 4-2
 Working 4-7
 Glossary definition A-28
CDC CONNECT 14-1
CDCNET batch station 6-39, 40
 Central Processing Units 6-71
 Central software support 12
CHANGE_BACKUP_LABEL_TYPE 12-6
CHANGE_CATALOG_ENTRY 4-20, 39
CHANGE_COMMAND_SEARCH_MODE 5-8, 12
CHANGE_CONNECTION_ATTRIBUTE 2-4; 13-24
CHANGE_FILE_ATTRIBUTE 4-55, 61
CHANGE_JOB_ATTRIBUTE 6-12
CHANGE_JOB_LIMIT 6-70; 8-23
CHANGE_LINK_ATTRIBUTE 10-1, 2
CHANGE_LINK_ATTRIBUTE_CHARGE sub ADMV 10-3
CHANGE_LINK_ATTRIBUTE_FAMILY sub ADMV 10-3
CHANGE_LINK_ATTRIBUTE_PASSWORD sub ADMV 10-3
CHANGE_LINK_ATTRIBUTE_PROJECT sub ADMV 10-3
CHANGE_LINK_ATTRIBUTE_USER sub ADMV 10-3
CHANGE_LOGIN_PASSWORD 3-5, 7
CHANGE_MESSAGE_LEVEL 6-65
CHANGE_NATURAL_LANGUAGE 6-65
CHANGE_OUTPUT_ATTRIBUTE 6-44
CHANGE_SCL_OPTION 3-26

- CHANGE_TAPE_LABEL_ATTRIBUTE 11-12, 16, 19, 26; 12-7, 8, 20
- CHANGE_TERM_CONN_DEFAULT 13-24
- CHANGE_TERMINAL_ATTRIBUTE 3-10, 12; 13-4
- CHANGE_UTILITY_ATTRIBUTES 9-66
- \$CHAR function 9-6
- Character
- Conversion (ANSI file) 11-11; D-18
 - Dollar sign in names 2-7, 27
 - Glossary definition A-3
 - Network command 3-12; 13-11
 - Use in names 2-6
- CHARACTER_CONVERSION
- File attribute 4-42; 11-11
 - Tape label attribute 11-19
- CHARACTER_FLOW_CONTROL terminal attribute 13-6
- Character set
- ANSI file 11-11; D-17
 - ASCII C-1
 - EBCDIC C-5
- CHARACTER_SET tape label attribute 11-20
- Child job 6-2
- Glossary definition A-4
- Child task 6-2
- Glossary definition A-4
- Christensen protocol 14-6
- COBOL, glossary definition A-4
- CODE_SET terminal attribute 13-7
- COLLATE_TABLE_NAME file attribute 4-42
- Collating sequence, glossary definition A-4
- COLLECT_TEXT 9-3, 12
- Combinations of expressions 7-44
- \$COMMAND 9-15
- Command
- Abbreviation 2-12
 - Definition 2-11
 - Descriptions 2-13
 - Execution 2-12
 - Format used in this manual 11
 - Glossary definition A-4
 - Inserting into command stream 9-17
 - Naming conventions 2-11
 - Suspending processing 8-26
 - Termination 8-20
 - Used in procedures 9-3
 - Utility 2-24
- COMMAND
- Command 9-52
 - Job file 6-57
- COMMAND_FAULT condition 8-20
- Command library, glossary definition A-4
- Command list 1-3; 5-1; 9-45
- Adding entries 5-8; 9-48
 - Changing 5-8, 11
 - Definition 2-13
 - Entries 5-2
 - Catalogs 5-4
 - Command utilities 5-5
 - Control commands 5-3, 6
 - Control statements 5-3, 5
 - Deleting 5-11
 - Displaying 5-13
 - Message modules 5-3
 - Moving 5-11
 - Object libraries 5-3
 - \$SYSTEM 5-4
 - Glossary definition A-4
 - Search modes 5-7
 - Changing 5-12
 - Searching 2-13
 - \$SYSTEM 2-13
- \$COMMAND_OF_CALLER 9-15, 53
- Command references 5-1
- \$COMMAND_SOURCE function 9-9
- Command stream
- \$COMMAND 9-15
 - \$COMMAND_OF_CALLER 9-15
 - Current position 9-15
 - Definition 2-2
 - Glossary definition A-4

- Inserting commands into 9-17
- Inserting files into 9-14
- Inserting strings into 9-14
- Structuring 8-3
- Command table 9-52, 58
- Command utility 5-5
 - Defining a 9-49
 - Definition 9-45
 - Glossary definition A-4
- COMMENT_BANNER
 - Job attribute 6-12
 - Output attribute 6-45
- Comment, glossary definition A-4
- Comments
 - About this manual 12
 - Formatter 9-70
 - in procedures 9-70
 - SCL 2-10; 9-23
- COMPARE_FILE 4-80
- Comparing files 4-80
- Compiler, glossary definition A-5
- COMPRESSION_PROCEDURE_NAME file attribute 4-43
- Concatenation operator 7-39
- Condition
 - Canceling 8-25
 - Categories 8-20
 - Codes 8-19
 - Handling 8-20
 - Processing 8-16
 - Severity levels 8-21
- \$CONDITION_CODE
 - function 9-8
- Condition code, glossary definition A-5
- Condition handler, glossary definition A-5
- \$CONDITION_NAME
 - function 9-8
- Condition name, glossary definition A-5
- CONDITION (status field) 2-23; 7-20, 32; 8-19
- Connecting files 6-59

- Connection attribute 13-1, 2
 - Attribute set 13-25
 - Changing 13-3
 - Default level 13-23
 - Changing 13-24
 - Displaying 13-24
 - Default values 13-34
 - File level 13-23
 - Changing 13-24
 - Displaying 13-24
 - Glossary definition A-5
 - Instance of open level 13-23
 - Network applicability 13-36
 - Editing mode 13-38
- Constants 2-29
- CONTINUE
 - Statement 8-23, 24
 - Syntax 8-24
- Control command 5-6
- Control command, glossary definition A-5
- CONTROL_FAMILY
 - Job attribute 6-13
 - Output attribute 6-45
- Control family, glossary definition A-5
- Control statement 5-5
 - Block 8-2
- BLOCK/BLOCKEND 8-5
- Condition handler 2-34
- Conditional execution 2-34
- CONTINUE 8-24
- CYCLE 8-10
 - Definition 2-34
- Descriptions 2-13
- EXIT 8-12
 - Flow control 2-34; 8-10
- FOR/FOREND 7-4; 8-9
 - Glossary definition A-5
- IF/IFEND 8-15
- LOOP/LOOPEND 8-6
- POP 8-28
- PUSH 8-27
- REPEAT/UNTIL 8-8
 - Structured 2-34; 8-3
- WHEN/WHENEND 8-22
- WHILE/WHILEEND 8-7
- CONTROL_USER
 - Job attribute 6-13
 - Output attribute 6-45

Control user, glossary definition

DELETE_CATALOG_CONTENT sub BACPF

Control user, glossary
definition A-5

Conventions

- Boolean 7-31
- Command names 2-11
- Expressions 2-33; 7-37
- Functions names 2-27
- Hexadecimal integers 7-24
- In this manual 11
- Parameter names 2-14
- Procedure format 9-76
- Procedure names 2-36
- Real number 7-27
- Variable names 7-3

COPIES

- Job attribute 6-13
- Output attribute 6-45

COPIES_PRINTED output
attribute 6-45

COPY_FILE 4-64

Copying files 4-64

- Between file
 - organizations 4-66
- Byte-addressable to
 - byte-addressable 4-73
- Keyed to keyed 4-74
- Keyed to sequential 4-77
- List files 4-79
- On dual-state systems 10-2
- Sequential to keyed 4-71
- Sequential to sequential 4-67
- Tape files 11-17, 18
- Using format effectors 6-54

CPI, glossary definition A-6

CPU_TIME_LIMIT job
attribute 6-13

CREATE_CATALOG 4-13

CREATE_CATALOG_
PERMIT 9-61

CREATE_COMMAND_LIST_
ENTRY 5-8; 9-48, 60, 61

CREATE_FILE 4-17, 31

CREATE_FILE_
CONNECTION 6-59

CREATE_FILE_PERMIT 9-60

CREATE_OBJECT_
LIBRARY 9-47

CREATE_REMOTE_
VALIDATION 4-84

CREATE_VARIABLE 2-30; 7-5;
8-18; 9-3

Creation date (ANSI file) 11-9;
D-12

CREATION_DATE tape label
attribute 11-20

CYBIL, glossary definition A-6

Cycle

- Descriptor 4-10, 19
- Glossary definition A-6

CYCLE

- Statement 8-10
- Syntax 8-10

Cycle number, glossary
definition A-6

CYCLIC_AGING_INTERVAL
job attribute 6-13

D

D type record, glossary
definition A-6

DATA_MODE output
attribute 6-45

DATA_PADDING file
attribute 4-43

Data type

- ANY 2-22; 7-36
- Application value 2-22; 7-36
- BOOLEAN 2-22, 31; 7-31
- FILE 2-21, 31; 7-34
- INTEGER 2-21, 31; 7-23
- KEY 7-35
- KEYWORD 2-22, 31
- NAME 2-21, 31; 7-35
- Procedure parameter 9-27
- REAL 2-22; 7-27
- STATUS 2-22; 7-32
- STRING 2-21, 31; 7-29
- UNION 7-36

\$DATE function 9-5

Decimal equivalents 7-47

Default, glossary definition A-6

DEFINE_PRIMARY_
TASK 6-37

DELETE_CATALOG 4-15

DELETE_CATALOG_CONTENT
sub BACPF 12-11, 17

- DELETE_CATALOG_
- PERMIT 4-26
- DELETE_COMMAND_LIST_
- ENTRY 5-3, 8, 11
- DELETE_FILE 4-18
- DELETE_FILE_
- CONNECTION 6-61
- DELETE_FILE_CONTENT sub
- BACPF 12-16, 17
- DELETE_FILE_PERMIT 4-26
- DELETE_REMOTE_
- VALIDATION 4-84
- DELETE_VARIABLE 7-19; 9-3
- Delimiter
- Apostrophe 7-29
 - Comma 2-15
 - Glossary definition A-6
 - Period 7-34
 - Semicolon 7-34; 9-24
 - Space 2-10
- Desktop/VE 14-1, 21
- DETACH_FILE 4-31; 9-10
- DETACH_JOB 3-16
- DETACHED_JOB_WAIT_TIME
- job attribute 3-19; 6-13
- DEVICE
- Job attribute 6-14
 - Output attribute 6-46
- DEVICE_TYPE output
- attribute 6-46
- Diagnostic messages 7-32
- Digit, glossary definition A-6
- Direct-access file organization, glossary definition A-7
- Directive, glossary definition A-7
- DISPATCHING_PRIORITY job
- attribute 6-14
- DISPLAY_ACTIVE_TASK 6-37
- DISPLAY_BACKUP_FILE sub
- RESPF 12-4, 21, 27
- DISPLAY_BACKUP_LABEL_
- TYPE 12-6
- DISPLAY_CATALOG 4-20, 26
- DISPLAY_CATALOG_
- ENTRY 4-19, 26, 39
- Display code, glossary definition A-7
- DISPLAY_COMMAND_
- INFORMATION 3-24; 9-65
- DISPLAY_COMMAND_
- LIST 5-12
- DISPLAY_COMMAND_LIST_
- ENTRY 2-26; 3-25; 9-65
- DISPLAY_CONNECTION_
- ATTRIBUTE 13-24
- DISPLAY_FILE 4-82
- DISPLAY_FILE_
- CONNECTION 6-61
- DISPLAY_FUNCTION_
- INFORMATION 3-24
- DISPLAY_JOB_
- ATTRIBUTE 6-12
- DISPLAY_JOB_ATTRIBUTE_
- DEFAULTS 6-12
- DISPLAY_JOB_HISTORY 6-65, 69
- DISPLAY_JOB_STATUS 6-32, 65
- DISPLAY_LINK_
- ATTRIBUTES 10-2
- DISPLAY_LOG 6-65, 67
- DISPLAY_MESSAGE 6-65
- DISPLAY_OUTPUT_
- ATTRIBUTE 6-44
- DISPLAY_OUTPUT_
- HISTORY 6-65, 69
- DISPLAY_OUTPUT_
- STATUS 6-52
- DISPLAY_REMOTE_
- VALIDATION 4-85
- DISPLAY_TAPE_LABEL_
- ATTRIBUTE 11-27
- DISPLAY_TERM_CONN_
- DEFAULT 13-24
- DISPLAY_TERMINAL_
- ATTRIBUTE 3-12; 13-4
- DISPLAY_VALUE 2-28; 7-46; 9-3
- DISPLAY_VARIABLE_
- LIST 7-19; 9-3
- Displaying values 7-46
- Documentation B-1
- Dollar sign character in names 2-7, 27
- Dual-state system 1-4; 10-1
- File
- Access 10-1, 2
 - Printing 10-1
 - Glossary definition A-7

E

- EARLIEST_PRINT_TIME
 - Job attribute 6-14
 - Output attribute 6-46
- EARLIEST_RUN_TIME job attribute 6-14
- EBCDIC character set C-5
- \$ECHO standard file 6-58
- ECHOPLEX terminal attribute 13-7
- EDIT_FILE utility 9-18, 47
- Ellipsis 2-5; 9-25
 - Glossary definition A-7
- ELSE clause 8-15
- ELSEIF clause 8-15
- EMBEDDED_KEY file attribute 4-43
- Embedded key, glossary definition A-7
- END_LINE_CHARACTER terminal attribute 13-7
- END_LINE_POSITIONING terminal attribute 13-8
- END_OF_INFORMATION connection attribute 13-26
- End-of-information, glossary definition A-7
- End-of-partition, glossary definition A-7
- END_OUTPUT_SEQUENCE terminal attribute 13-8
- END_PAGE_ACTION terminal attribute 13-8
- END_PARTIAL_CHARACTER terminal attribute 13-9
- END_PARTIAL_POSITIONING terminal attribute 13-9
- Environment
 - Changing 8-27
 - Objects 8-26
 - Restoring 8-28
- Environment objects
 - Glossary definition A-8
- EOF1 tape label 11-13; D-19
- EOF2 tape label 11-13; D-21
- EOF3 through EOF9 tape labels D-25
- \$EOI 4-12
- EOV1 tape label D-22
- EOV2 tape label D-24
- Epilog 6-9
 - Glossary definition A-8
- Error
 - Conditions 8-19
 - Processing 2-23; 8-17
 - Semantic 8-20
 - Syntax 8-20
- ERROR_EXIT_PROCEDURE_NAME file attribute 4-43
- ERROR_LIMIT file attribute 4-43
- Error processing
 - SCL formatter 9-80
 - Syntax 9-16
- \$ERRORS standard file 6-58
- ESTIMATED_RECORD_COUNT file attribute 4-44
- Examples online manual 3-31
- EXCLUDE_CATALOG sub BACPF 12-10, 17
- EXCLUDE_FILE sub BACPF 12-13, 17
- EXCLUDE_HIGHEST_CYCLE sub BACPF 12-5, 16, 17
- EXCLUSIVE search mode 5-7
- EXECUTE_COMMAND 5-1; 6-34, 35
- EXECUTE_TASK 6-34
- Executing
 - Commands 5-1
 - SCL procedures 5-1
- Execution ring, glossary definition A-8
- EXIT
 - Statement 8-5, 6, 12; 9-15
 - Syntax 8-12
- Expedited data 3-3
- Expiration
 - Date (ANSI file) 11-9; D-13
 - Date (Password) 3-7
 - Interval (Password) 3-7
 - Warning interval (Password) 3-7
- EXPIRATION_DATE tape label attribute 11-20
- EXPLAIN 3-28

Explicit

- Command execution 5-4
- Variable creation 2-30; 7-5, 17
- Variable deletion 7-19

Exponent 7-27**Expression**

- As parameter value 2-20
- Conventions 2-33; 7-37
- Definition 2-33; 7-37
- Displaying the value of 7-46
- Glossary definition A-8
- Integer 7-38
- Logical 7-40
- Operators 7-37
- Relational 7-41
- String 7-39
- Syntax of G-3
- Use of spaces in 2-33
- Valid combinations 7-44

EXTERNAL_

- CHARACTERISTICS job attribute 6-14

EXTERNAL_

- CHARACTERISTICS output attribute 6-46

F

- F type record, glossary definition A-8

- Family administrator, glossary definition A-8

- Family, glossary definition A-8

- Family name 3-5

- Glossary definition A-8

- Family path, glossary definition A-9

- Field, glossary definition A-9

File

- Accessibility code (ANSI file) 11-9; D-14
- Data type 2-21, 31; 7-34
- Identifier (ANSI file) 11-8; D-9
- Section identifier (ANSI file) 11-8
- Section number (ANSI file) 11-8; D-10

Set 11-6

- Set identifier (ANSI file) 11-8; D-9

- FILE_ACCESS_PROCEDURE_NAME file attribute 4-44

- FILE_ACCESSIBILITY_CODE tape label attribute 11-20

- File and catalog structure 4-1

File attribute

- Attribute set 4-40
- Changable attribute 4-55
- Changing 4-61
- Displaying 4-62
- Dual-state files 10-2
- Establishing for a new file 4-58
- Glossary definition A-9
- Keyed files 4-60
- Preserved 4-40
- Preserving on dual-state transfers 10-4
- Record access files 4-59
- Temporary 4-40
- Using CONNECT 14-1

- File connection, glossary definition A-9

- FILE_CONTENTS file attribute 4-44; 5-4; 13-48

- \$FILE function 9-6

- FILE_IDENTIFIER tape label attribute 11-20

- FILE_LABEL_TYPE file attribute 4-45

- FILE_LIMIT file attribute 4-45

- File name, glossary definition A-9

- FILE_ORGANIZATION file attribute 4-46

- File organization, glossary definition A-9

- File position, glossary definition A-10

- FILE_POSITION output attribute 6-46

- FILE_PROCESSOR file attribute 4-46; 5-4

File reference 7-34

- Element 7-34
- Glossary definition A-10
- Maximum length 7-34

- FILE_SEQUENCE_NUMBER
tape label attribute 11-20
- FILE_SET_IDENTIFIER tape
label attribute 11-21
- FILE_SET_POSITION tape
label attribute 11-21
- FILE_SIZE output
attribute 6-46
- FILE_STRUCTURE file
attribute 4-47; 5-4
- File transfer
KERMIT 14-18
XMODEM 14-1
- Files 4-1
Access control entries 4-22
Multiple entries 4-26
Attaching and detaching 4-30
Backing up 12-1, 12
Byte-addressable organization,
glossary definition A-3
COMMAND job file 6-57
Comparing 4-80
Connections 6-59
Creating 6-59
Deleting 6-61
Displaying 6-61
Copying 4-64
Creating 4-13, 15
Explicit 4-17
Implicit 4-16
CYBER binary files 14-7
Cycles 4-6, 10; 12-13
Glossary definition A-6
Number, glossary
definition A-6
Deleting 4-18
Detaching 4-30
Direct-access organization,
glossary definition A-7
Displaying 12-9
Dual-state 10-1
Copying 10-2
Glossary definition A-9
Indexed-sequential
organization, glossary
definition A-11
INPUT job file 6-57; 13-1
Inserting into command
stream 9-14
Job 6-57
- \$JOB_LOG job file 6-57
Micro binary 14-6
Microcomputer to NOS/VE
transfers 14-1
Names 4-5
Open position 4-6, 12
\$ASIS 4-38
\$BOI 4-12
\$EOI 4-12
Output file 6-44
OUTPUT job file 6-38, 57;
13-1
Passwords 4-39
Paths 1-2; 4-5, 8, 14
Permanent, glossary
definition A-18
Permits 4-21
Positioning 4-12
Printing 6-38
At CDCNET Batch
Stations 6-40
At Remote and partner
systems 6-41
Using format effectors 6-56
Referencing 1-2; 4-5, 8
Remote management 4-84
Explicit 4-86
Implicit 4-85
Restoring 12-1, 19, 25
Sharing 4-21, 33
Standard 6-57
Standard file 6-58
Glossary definition A-23
Tapes 11-1
ANSI-labelled 11-5; D-1
Multifile 11-12
Requesting 11-4
Rewinding 11-5
Unlabelled 11-5, 32
Temporary 4-2
Text files 14-7
Unreadable data 12-9
Flow control statement 8-10
\$FNAME function 9-5
FOLD_LINE terminal
attribute 13-10
FOR/FOREND
Statement 7-4; 8-9
Syntax 8-9

FORCED_WRITE file attribute 4-47
FORM_FEED_DELAY terminal attribute 13-10
FORM_FEED_SEQUENCE terminal attribute 13-10
 Format effectors 6-53; 9-13; E-1
 With **COPY_FILE** 6-54
 With **PRINT_FILE** 6-56
FORMAT_SCL_PROC 9-3, 67
 Formatting
 Conventions for procedures 9-76
 Procedures 9-19, 67
 Utilities 9-47, 76
FORMS_CODE
 Job attribute 6-15
 Output attribute 6-46
FORTRAN, glossary definition A-10
 Function
 Definition 2-26
 Descriptions 2-13
 Displaying values 2-28
 Execution 2-27
 Glossary definition A-11
 Naming conventions 2-27
 Parameters 2-27
 Use of spaces in 2-27
 Used in procedures 9-3
 Utility 2-26

G

Generation
 Number (ANSI file) 11-8; D-11
 Version number (ANSI file) 11-8; D-11
GENERATION_NUMBER tape label attribute 11-22
GENERATION_VERSION_NUMBER tape label attribute 11-22
GET_FILE 10-2
GLOBAL search mode 5-7

H

Hashing procedure, glossary definition A-11
HASHING_PROCEDURE_NAME file attribute 4-47
HDR1 tape label 11-8; D-8
 Processing 11-13
HDR2 tape label 11-9; D-9
 NOS/VE extensions 11-10
 Processing 11-13
HDR3 through **HDR9** tape labels D-25
HELP 2-26; 3-25
 Hexadecimal integer
 Conventions 7-24
 Equivalents 7-47
 Hierarchy of catalogs 1-2; 4-3
 High order bit 14-9
HOLD_PAGE_OVER terminal attribute 13-11, 49
HOLD_PAGE terminal attribute 13-10, 48
 Home block, glossary definition A-11
 Hotline 12

I

IF/IFEND
 Statement 8-15
 Syntax 8-15
 Implicit
 File attach 9-10
 File detach 9-10
 Variable creation 2-30; 7-4, 16
INCLUDE_COMMAND 9-4, 17
INCLUDE_CYCLE sub BACPF 12-13, 17
INCLUDE_EMPTY_CATALOG sub BACPF 12-11, 18
INCLUDE_FILE
 Block 8-2, 17; 9-14
 Command 9-4, 14, 53
 Using **\$COMMAND** 9-15
 Using **\$COMMAND_OF_CALLER** 9-15

INCLUDE_LARGE_CYCLE sub
 BACPF 12-14, 18
 INCLUDE_LINE
 Block 8-2, 17; 9-16
 Command 9-4, 16
 INCLUDE_SMALL_CYCLE sub
 BACPF 12-14, 18
 INDEX_LEVELS file
 attribute 4-47
 INDEX_PADDING file
 attribute 4-48
 Indexed-sequential file
 organization, glossary
 definition A-11
 INITIAL_HOME_BLOCK_
 COUNT file attribute 4-48
 Initiating tasks 6-35
 Input
 Attributes 6-4
 Blocks 8-17
 Current source of 9-15
 Data 13-41
 Definition 2-2
 Delimiter 8-4
 Delimiters 2-10, 14, 20, 27,
 33; 7-20, 21, 29, 33; 9-22
 Lines 2-5
 Maximum line length 2-5
 Multiple statements 2-5
 Typed-ahead 13-45
 INPUT_BLOCK_SIZE
 connection attribute 13-26
 INPUT_EDITING_MODE
 connection attribute 3-9, 12,
 14; 13-27, 42, 46
 INPUT job file 6-57; 9-10; 13-1
 Input lines, glossary
 definition A-11
 INPUT_OUTPUT_MODE
 connection attribute 13-27
 \$INPUT standard file 6-58;
 9-10
 INPUT_TIMEOUT connection
 attribute 13-28
 INPUT_TIMEOUT_LENGTH
 connection attribute 13-29
 INPUT_TIMEOUT_PURGE
 connection attribute 13-29

Integer
 Data type 2-21, 31; 7-23
 Definition 7-23
 Expressions 7-38
 Glossary definition A-11
 Hexadecimal conventions 7-24
 Operators 7-38
 Radix specification 7-23
 Signed 7-25
 Integer constant, glossary
 definition A-11
 \$INTEGER function 9-5
 \$INTERACTION_STYLE
 function 9-7
 Interaction style, glossary
 definition A-12
 Interactive
 Input 3-9
 Jobs 6-1
 Procedures 9-15
 Sessions 3-1
 Utilities 9-15
 Interactive job
 Input 2-2, 4
 Prompt 2-4
 Interactive mode, glossary
 definition A-12
 INTERNAL_CODE file
 attribute 4-48
 Interpreter, SCL 2-1
 INTERRUPT condition 8-20
 Interrupting the system 3-13
 In screen mode 3-16
 Pause break 3-15
 RESUME_
 COMMAND 3-15
 TERMINATE_
 COMMAND 3-15
 Terminate break 3-16
 Iteration of repetitive
 statement 8-10

J

\$JOB 6-72
 Job
 Block 2-4; 7-2, 13; 8-2, 17
 File 6-57; 9-10
 History log 6-66

- Limits 3-20; 6-70
- JOB_ABORT_DISPOSITION job attribute 6-15
- Job attribute
 - Attribute set 6-12
 - Changing 6-12
 - Displaying 6-12
 - Glossary definition A-12
- Job class, glossary definition A-12
- JOB_CLASS job attribute 6-15
- Job classes
 - Assignment 6-4
 - Example
 - Batch jobs 6-5
 - Interactive jobs 6-7
 - Restrictions 6-4
 - Selection 6-4
 - UNASSIGNED 6-6
 - Validation 6-4
- \$JOB_DEFAULTS function 9-7
- Job file 6-57; 9-10
 - Glossary definition A-12
- \$JOB function 9-7
- Job history log 6-66
 - Displaying 6-69
- JOB/JOBEND 6-29
- \$JOB_LIMIT function 9-7
- Job limits 3-20; 6-70
- Job limits, glossary definition A-12
- Job log, glossary definition A-13
- \$JOB_LOG job file 6-57
- JOB_MODE job attribute 6-15
- \$JOB_OUTPUT function 9-8
- JOB_QUALIFIER job attribute 6-15
- JOB_RECOVERY_DISPOSITION job attribute 6-16
- JOB (Scope of variable) 7-12
- JOB_SIZE job attribute 6-16
- \$JOB_STATUS function 9-7
- JOB_SUBMISSION_TIME job attribute 6-16

- Jobs
 - Attributes 1-3; 6-12
 - Batch 1-1; 6-1, 25
 - Glossary definition A-2
 - I/O stations 6-31
 - Input 2-2
 - JOB/JOBEND 6-29
 - SUBMIT_JOB 6-26
 - Submitting 6-25, 29
 - Submitting to remote systems 6-26
 - Termination 8-20
 - Child 6-2
 - Glossary definition A-4
 - Classes 6-3
 - Environments 1-3; 6-8
 - Epilogs 6-9
 - Changing names 6-11
 - Glossary definition A-12
 - Input attributes 6-4
 - Interactive
 - Input 2-2, 4; 3-9
 - Jobs 1-1; 3-1; 6-1
 - Prompt 2-4
 - Logging in and out 3-4
 - Logs 6-64
 - Message placement 6-65
 - Message level 6-65
 - Multiple file attachments 4-36
 - Names 6-2
 - Natural languages 6-65
 - Output 6-38
 - Output line 13-46
 - Overview 1-1; 6-1
 - Parent 6-2
 - Glossary definition A-18
 - Prologs 6-9
 - Changing names 6-11
 - Terminating 6-32

K

- KERMIT-VE 14-1, 18
- KERMIT-VE file transfers 14-18
- Key, glossary definition A-13
- KEY_LENGTH file attribute 4-49

KEY_POSITION file
 attribute 4-49
 KEY_TYPE file attribute 4-49
 Keyed file
 Copying 4-74, 77
 File attributes 4-60
 Glossary definition A-13
 Keyword
 Data type 2-22, 31; 7-35
 Definition 7-35
 Glossary definition A-13

L

Label
 Glossary definition A-13
 Identifier D-16
 LATEST_PRINT_TIME
 Job attribute 6-16
 Output attribute 6-47
 LATEST_RUN_TIME job
 attribute 6-16
 LIMIT_FAULT condition 8-20
 Limits 3-20; 6-70
 LINE_FEED_DELAY terminal
 attribute 13-11
 LINE_FEED_SEQUENCE
 terminal attribute 13-11
 Line mode
 Glossary definition A-13
 Processing 3-9
 Line mode interaction, glossary
 definition A-13
 LINE_NUMBER file
 attribute 4-49
 Link attribute 10-2
 Changing 10-2, 3
 Displaying 10-2
 Glossary definition A-13
 List, glossary definition A-13
 \$LIST standard file 6-58
 Load module, glossary
 definition A-14
 \$LOCAL catalog 4-2
 Local file 4-2
 Glossary definition A-14
 Local path, glossary
 definition A-14
 LOCAL (Scope of variable) 7-12

Local system, glossary
 definition A-14
 LOCK_EXPIRATION_TIME file
 attribute 4-49
 Lock, glossary definition A-14
 LOG_RESIDENCE file
 attribute 4-50
 Logging in and out 3-4
 LOGGING_OPTIONS file
 attribute 4-50
 Logical
 Expressions 7-40
 Operators 7-40
 LOGIN_ACCOUNT
 Job attribute 6-16
 Output attribute 6-47
 LOGIN_FAMILY
 Job attribute 6-17
 Output attribute 6-47
 Login Information 3-5
 For NAMVE/CDCNET 6-8
 Validation level 3-6
 LOGIN_PROJECT
 Job attribute 6-17
 Output attribute 6-47
 LOGIN_USER
 Job attribute 6-17
 Output attribute 6-47
 Login user, glossary
 definition A-14
 Logs 6-64
 Display message area 6-66
 Displaying 6-65
 Message placement 6-65
 LOOP/LOOPEND
 Exiting 8-6
 Statement 8-6
 Syntax 8-6
 Lower bound (array) 7-6

M

MAGNETIC_TAPE_LIMIT job
 attribute 6-17
 \$MAINFRAME 6-71
 Mainframe attribute, glossary
 definition A-14
 \$MAINFRAME function 9-7

MANAGE_REMOTE_FILES 4-86
 Mantissa 7-27
 Manual set for NOS/VE 10
 Manuals B-1
 Mass storage, glossary definition A-14
 Master catalog, glossary definition A-15
 \$MAX_INTEGER function 2-29; 7-23; 9-5
 \$MAX_NAME function 2-29; 9-5
 \$MAX_STRING function 2-29; 9-5
 \$MAX_VALUE_SETS function 2-29; 9-5, 34
 \$MAX_VALUES function 2-29; 9-5
 MAXIMUM_BLOCK_LENGTH
 File attribute 4-50
 Tape label attribute 11-22
 Maximum expiration interval (Password) 3-7
 Maximum length
 Catalog reference 7-34
 File reference 7-34
 Input lines 2-5
 SCL names 2-6
 Strings 7-29
 MAXIMUM_RECORD_LENGTH
 File attribute 4-50
 Tape label attribute 11-23
 MAXIMUM_WORKING_SET
 job attribute 6-17
 Message
 Language 6-65
 Level 6-65
 MESSAGE_CONTROL file attribute 4-51
 Message level
 Changing 6-65
 \$MESSAGE_LEVEL function 9-7
 Metalanguage symbols F-1
 Micro file transfer 14-1
 \$MIN_INTEGER function 2-29; 7-23; 9-5
 MINIMUM_BLOCK_LENGTH
 file attribute 4-51

MINIMUM_RECORD_LENGTH
 file attribute 4-51
 MINIMUM_WORKING_SET
 job attribute 6-17
 Minus sign 7-25
 \$MOD function 9-9
 Module, glossary definition A-15
 Multifile set, glossary definition A-15
 Multifile tape volumes 11-12
 Multiple statements on input line 2-5
 Multiprocessing 6-71

N

Name
 Allowable characters 2-6
 Data type 2-21, 31; 7-35
 Definition 2-6
 Procedure parameter 9-26
 Procedures 9-22
 Reserved 2-9
 Scope of variable 7-12
 Special characters 2-7
 Upper and lowercase characters 2-8
 Use of dollar sign character 2-7, 27
 \$NAME function 9-5
 Natural language 6-65
 Changing 6-65
 \$NATURAL_LANGUAGE function 9-7
 Natural language, glossary definition A-15
 Nested procedure call 9-20
 Network
 Command characters 3-12; 13-11
 Keystroke sequence 3-12
 Network access method, glossary definition A-15
 Network command character, glossary definition A-15
 NETWORK_COMMAND_CHARACTER terminal attribute 13-11

Networks 1-4; 3-2; 13-1, 3
 Communication Line 3-3
 Data Paths 3-3
 Service 3-3
 Standard 3-4
 Terminal Interface
 Program 3-2
 User-controlled
 interrupts 3-13
 Screen mode 3-16
 Nonembedded key, glossary
 definition A-16
 NORMAL (status field) 2-23;
 7-20, 32
 NOS
 Files 10-1
 Glossary definition A-16
 NOS/BE
 Files 10-1
 Glossary definition A-16
 NOS/VE
 Glossary definition A-16
 Manual set 10
 NTF system, glossary
 definition A-16
 \$NULL standard file 6-58
 Numeric character
 Glossary definition A-16
 Use in SCL names 2-6

O

Object code, glossary
 definition A-16
 Object file, glossary
 definition A-16
 Object library 5-3, 10
 Creating 5-10
 Generating 9-48
 Glossary definition A-17
 Object module, glossary
 definition A-17
 Octal equivalents 7-47
 Online help 1-4; 3-24, 28
 Online manuals 3-28; B-1
 Examples manual 3-31
 Line mode operation 3-30
 Screen mode operation 3-28

Open operation, glossary
 definition A-17
 OPEN_POSITION file
 attribute 4-51
 Operand and operator
 combinations 7-44
 Operand, glossary
 definition A-17
 Operator
 Boolean expressions 7-40
 Glossary definition A-17
 Integer expressions 7-38
 Logical expressions 7-40
 Relational expressions 7-41
 String expressions 7-39
 OPERATOR_FAMILY
 Job attribute 6-18
 Output attribute 6-47
 OPERATOR_USER
 Job attribute 6-18
 Output attribute 6-47
 \$OPTIONAL 9-40
 \$ORD function 9-6
 Order dependence of
 parameters 2-15
 Order of operations 7-37
 Ordering manuals B-1
 ORIGINATING_APPLICATION_
 NAME
 Job attribute 6-18
 Output attribute 6-48
 OSV\$COMMAND variable 8-22
 OSV\$STATUS variable 7-19;
 8-22
 Output attribute
 Attribute set 6-45
 Changing 6-44
 Displaying 6-44
 Output block 13-46
 OUTPUT_CLASS
 Job attribute 6-18
 Output attribute 6-48
 OUTPUT_DESTINATION
 Job attribute 6-18
 Output attribute 6-48
 OUTPUT_DESTINATION_
 USAGE
 Job attribute 6-19
 Output attribute 6-48

OUTPUT_DISPOSITION job
 attribute 6-20
 Output file 6-44
 Deleting 6-52
 Displaying its status 6-52
 Format effectors 6-53
 Printing on partner
 systems 10-1
 Using the wait queue 6-52
 OUTPUT job file 6-57; 13-1
 Output line 13-46
 Interline positioning 13-48
 Page holding 13-48
 Output message 13-46
 OUTPUT_PRIORITY
 Job attribute 6-21
 Output attribute 6-49
 Output queue 6-39
 Deleting files 6-52
 Output record 13-46
 \$OUTPUT standard file 6-58
 \$OUTPUT_STATUS
 function 9-8
 OUTPUT_SUBMISSION_TIME
 output attribute 6-49
 Owner identifier 11-7

P

PADDING_CHARACTER
 File attribute 4-52
 Tape label attribute 11-23
 Padding character (ANSI
 file) 11-11; D-17
 Padding, glossary
 definition A-17
 PAGE_AGING_INTERVAL job
 attribute 6-21
 PAGE_FORMAT file
 attribute 4-52
 Page length 13-50
 PAGE_LENGTH
 file attribute 13-50
 File attribute 4-53
 Terminal attribute 13-11, 50
 Page width 13-50
 PAGE_WIDTH
 File attribute 4-53; 13-50
 Terminal attribute 13-12, 50

Parameter
 Abbreviation 2-16
 Definition 2-14
 Glossary definition A-17
 Naming conventions 2-14
 Position-dependent 2-15
 Position-independent 2-15
 Prompting 1-4; 9-64
 Use of spaces in 2-14
 Value list 2-17
 Parameter definition
 Procedure 9-24
 Parameter expression, glossary
 definition A-18
 \$PARAMETER function 9-8
 \$PARAMETER_LIST
 function 9-8
 Parameter list, glossary
 definition A-18
 Parameter name, glossary
 definition A-18
 Parameter prompting 3-26
 Parameter value
 Expressions 2-20
 Types 2-21
 Use of spaces in 2-20
 Parent job 6-2
 Glossary definition A-18
 Parent task 6-2
 Glossary definition A-18
 PARITY terminal
 attribute 13-12
 PARTIAL_CHARACTER_
 FORWARDING connection
 attribute 13-29
 Partner system, glossary
 definition A-18
 Password 3-5, 7
 Changing 3-7
 Expiration date 3-7
 Expiration interval 3-7
 Expiration warning
 interval 3-7
 Path 4-14
 Glossary definition A-18
 \$PATH function 9-6
 Pause break 3-15
 Glossary definition A-18
 RESUME_COMMAND 3-15
 Screen mode 3-16

- TERMINATE_
 - COMMAND 3-15
- PAUSE_BREAK_CHARACTER
 - terminal attribute 13-12
- Permanent catalog 4-3
 - Glossary definition A-18
- Permanent catalog hierarchy 1-2; 4-3
- Permanent file, glossary definition A-18
- Permanent File Transfer Facility 4-84
- Permit groups 4-22
- Plus sign 7-25
- POP statement 8-28
- Position-dependent parameter, glossary definition A-19
- Position-independent parameter, glossary definition A-19
- Positioning files 4-12
- Preserving NOS/VE file attributes 10-4
- PRESET_VALUE file attribute 4-53
- \$PREVIOUS_STATUS function 9-8
- Primary key, glossary definition A-19
- Primary task 6-37
 - Glossary definition A-19
- PRINT_FILE 6-38, 39
- Printing dual-state files 10-1
- Private reader 4-37
 - Glossary definition A-19
- Problem-reporting 12
- PROC/PROCEND
 - Statement 9-21
 - Syntax 9-21
- Procedure
 - Array specification 9-32
 - Block 7-2; 8-2, 17; 9-23
 - Commands used in 9-3
 - Definition 2-36
 - Example 9-43
 - Execution 9-19
 - Exiting from 8-14
 - Format 9-21
 - Formatting 9-19, 67
 - Functions used in 9-3
 - Glossary definition A-19
 - Interactive input 9-15
 - Names 9-22
 - Naming conventions 2-36
 - Nested call 9-20
 - Nested definition 9-20
 - Optional parameter 9-40
 - Parameter definition 9-24
 - Parameters 2-36
 - Recursive call 9-20
 - Required parameter 9-40
 - Syntax G-1
 - Variable specification 9-32
 - Writing 9-18
- Procedure library, glossary definition A-19
- Procedure parameter
 - ARRAY OF type 9-32
 - Data type 9-27
 - Default specification 9-40
 - Definition 9-24
 - INTEGER type 9-31
 - KEY type 9-31
 - LIST type 9-34
 - NAME type 9-29
 - Names 9-26
 - Optional 9-40
 - Range of values 9-34
 - Required 9-40
 - STRING type 9-30
 - Syntax 9-24
 - Value count 9-34
 - Value list 9-41
 - Value set count 9-34
 - Value specification 9-26
 - VAR OF type 9-32
- Processor attribute, glossary definition A-19
- \$PROCESSOR function 9-7
- Program attribute, glossary definition A-19
- Program description, glossary definition A-19
- PROGRAM_FAULT condition 8-20
- \$PROGRAM function 9-7
- Project name 3-5
- Prolog 6-8, 9
 - Glossary definition A-19

- Prompt
 String 2-4
 System 2-4
 UTILITY/UTILITYEND 9-51
 PROMPT_FILE connection
 attribute 13-30
 PROMPT_STRING connection
 attribute 2-4; 13-30
 Prompting for parameters 9-64
 PTF 4-84
 PURGE_DELAY
 Job attribute 6-21
 Output attribute 6-49
 PUSH statement 8-27
 PUT_LINE 9-4, 12
- Q**
- QTF, glossary definition A-20
 \$QUEUE function 9-9
 Queued-file transfer
 application 6-26
 \$QUOTE function 9-6
- R**
- Radix
 Glossary definition A-20
 Specification 7-24
 Random access, glossary
 definition A-20
 \$RANGE function 9-8, 36
 Range, glossary definition A-20
 Reading lines from a file 9-10
 Real (number)
 Conventions 7-27
 Data type 2-22; 7-27
 Definition 7-27
 Exponent 7-27
 Glossary definition A-20
 Mantissa 7-27
 Restrictions 7-27
 RECEIVE_FILE sub
 MANRF 4-86
 Record access files
 File attributes 4-59
 Record format (ANSI file) 11-9;
 D-16
 Record length (ANSI file) 11-10;
 D-16
 Record length extension (ANSI
 file) D-17
 RECORD_LIMIT file
 attribute 4-53
 RECORD_TYPE
 File attribute 4-53; 11-11
 Tape label attribute 11-23
 Record type (ANSI file) 11-11;
 D-17
 RECORDS_PER_BLOCK file
 attribute 4-54
 Recursive procedure call 9-20
 Reestablishing terminal
 connections 3-19
 Referencing
 Commands 5-1
 Files 4-5, 8
 Relational
 Expressions 7-41
 Operators 7-41
 Relative path, glossary
 definition A-20
 RELEASE_RESOURCE 11-3
 REMOVE_HOST_DIRECTIVE
 Dual-state systems 10-1
 Job attribute 6-22
 On PRINT_FILE 6-42
 On SUBMIT_JOB 6-27
 Output attribute 6-50
 Remote system, glossary
 definition A-20
 Remote validation 4-84
 Creating 4-84
 Deleting 4-84
 Displaying 4-84
 Glossary definition A-21
 \$REMOVE_VALIDATION
 function 9-7
 REPEAT/UNTIL
 Statement 8-8
 Syntax 8-8
 REPLACE_FILE 10-2
 REQUEST_MAGNETIC_
 TAPE 11-4; 12-5, 20
 REQUEST_TERMINAL 6-55;
 13-1
 \$REQUIRED 9-40
 RESERVE_RESOURCE 11-2

Residence of variables 7-2
 Resource limits 3-20; 6-70
 \$RESPONSE standard file 6-58
 RESTORE_ALL_FILES sub
 RESPF 12-25, 27
 RESTORE_CATALOG sub
 RESPF 12-23, 24, 27
 RESTORE_EXISTING_CATALOG sub RESPF 12-25, 27
 RESTORE_EXISTING_FILE sub RESPF 12-26, 27
 RESTORE_FILE sub
 RESPF 12-26, 27
 RESTORE_PERMANENT_FILES 12-1, 19
 Restore utility 12-1
 Displaying backup files 12-21
 From disk 12-20
 From tapes 12-20
 List file 12-4
 Restoring catalogs 12-23
 Existing 12-25
 Nonexisting 12-24
 Restoring files 12-25
 Existing 12-26
 Nonexisting 12-26
 Rules for file access 12-3, 19
 Starting the utility 12-19
 Stopping the utility 12-19
 RESTRICTED search mode 5-7
 RESUME_COMMAND 3-15
 RETRY clause 8-23, 24
 REWIND_FILE 11-5
 REWRITE_LABELS tape label attribute 11-23
 Ring attribute, glossary definition A-21
 \$RING function 9-7
 Ring, glossary definition A-21
 ROUTE_JOB 6-31
 ROUTING_BANNER
 Job attribute 6-22
 Output attribute 6-50
 Routing jobs
 From batch station input device 6-31
 From NOS 6-30
 From NOS/BE 6-30

S

S type record, glossary definition A-21
 \$SCAN_ANY function 9-6
 \$SCAN_NOT_ANY function 9-6
 \$SCAN_STRING function 9-6
 SCL
 Blocks 2-4
 Command list 5-2
 Comments 9-23
 Definition 2-1
 Formatter 9-19, 47, 67
 Glossary definition A-24
 Input 2-2, 5
 Interpreter 2-1, 3
 Names 2-6
 Prompts 2-4
 Statement 2-2
 Syntax F-1
 SCL name, glossary definition A-15
 Scope
 Example 7-13
 JOB 7-12
 LOCAL 7-12
 Name 7-12
 Of variables 7-2, 12
 XDCL 7-12
 XREF 7-12
 Screen mode
 Glossary definition A-21
 Processing 3-9
 Screen mode interaction, glossary definition A-21
 SCU, glossary definition A-22
 SEND_FILE sub MANRF 4-86
 Sense switches 6-72
 SENSE_SWITCHES job attribute 6-22
 Sequential access, glossary definition A-21
 Sequential file organization, glossary definition A-22
 Service 3-3
 Glossary definition A-22
 Service class, glossary definition A-22

- SERVICE_CLASS job attribute 6-23
- SET_BACKUP_OPTION sub
 - BACPF 12-9, 18
- \$SET_COUNT function 9-8, 35
- SET_FILE_ATTRIBUTES 4-58
- SET_LIST_OPTION sub
 - BACPF 12-4, 18
- SET_LIST_OPTION sub
 - RESPF 12-4, 27
- SET_MULTIPROCESSING_OPTIONS 6-71
- SET_SENSE_SWITCH 6-72
- SET_WORKING_CATALOG 4-7
- \$SEVERITY function 9-8
- Share modes (requirements) 4-24
- Sharing catalogs and files 4-21
- Sign
 - Definition 7-25
 - Glossary definition A-22
- Site administrator, glossary definition A-22
- SITE_INFORMATION job attribute 6-23
- SITE_INFORMATION output attribute 6-50
- SKIP_TAPE_MARK 11-34
- SOLVER 12
- Source code, glossary definition A-22
- Source library, glossary definition A-22
- Spaces
 - As delimiters 2-10
 - Definition 2-10
 - Use in array references 7-20
 - Use in expressions 2-33
 - Use in functions 2-27
 - Use in parameter values 2-20
 - Use in status references 7-21, 33
 - Use with parameters 2-14
 - Used in labelled statements 8-4
 - Used in procedure headers 9-22
- Special characters
 - In names 2-7
 - XMODEM 14-8
- \$SPECIFIED function 9-8, 41
- SRU, glossary definition A-25
- SRU_LIMIT job attribute 6-23
- Standard file 6-57; 9-10
 - Glossary definition A-23
- Standard NOS/VE networks 3-4
- Standard output, glossary definition A-23
- Statement
 - Assignment 2-32; 7-7
 - Continuation 2-5
 - Control 2-34
 - Definition 2-2
 - Delimiter 2-5
 - Examples 2-35
 - Glossary definition A-23
 - Multiple 2-5
- STATEMENT_INDENTIFIER file attribute 4-54
- Statement list
 - As a block 8-5
 - Conditional execution 8-15
 - Controlled repetition 8-9
 - Glossary definition A-23
 - Iteration 8-10
 - Postconditional repetition 8-8
 - Preconditional repetition 8-7
 - Unlimited repetition 8-6
- Statically linked block 7-10
 - Glossary definition A-23
- STATION
 - Job attribute 6-23
 - Output attribute 6-50
- Station, glossary definition A-23
- Status
 - Data type 2-22; 7-32
 - Definition 7-32
 - Fields 2-23; 7-20, 32; 8-19
 - Parameter 2-23; 8-17
 - References 7-20
- STATUS_ACTION terminal attribute 13-13
- \$STATUS function 9-8
- Status variable, glossary definition A-23
- STORE_BACKSPACE_CHARACTER connection attribute 13-30
- STORE_NULS_DELS connection attribute 13-30

String

- Concatenation 7-39
- Current length 7-29
- Data type 2-21, 31; 7-29
- Definition 7-29
- Delimiters 7-29
- Expressions 7-39
- Glossary definition A-24
- Inserting into command stream 9-14
- Maximum length 7-29
- Operators 7-39
- String constant, glossary definition A-24
- \$STRING function 9-5
- String length, glossary definition A-24
- \$STRLEN function 9-6
- \$STRREP function 9-5, 44
- Structured statement
 - Exiting from 8-12
 - Iteration 8-10
 - Labelled 8-3
 - Syntax 8-3
- Structured statement, glossary definition A-24
- Subcatalog, glossary definition A-24
- Subcommand 9-45
 - Definition 2-24; 9-52
 - Descriptions 2-13
 - Glossary definition A-24
 - Organization 9-58
 - Processors 9-52, 54
- Subject file 6-59
- SUBMIT_JOB 6-26
- \$SUBSTR function 9-6
- Synchronous tasks 6-33
 - Glossary definition A-24
- Syntax
 - Procedure G-1
 - SCL F-1
- \$SYSTEM 5-4
 - Command list 2-13
 - Glossary definition A-24
- System
 - Code (ANSI file) 11-9; D-14
 - Environments 8-26
 - Prompt 2-4
 - Variables 8-22

- System Command Language 2-1
 - Glossary definition A-24
 - See also SCL
- SYSTEM_FILE_NAME output attribute 6-50
- SYSTEM_JOB_NAME
 - Job attribute 6-23
 - Output attribute 6-50
- System operator, glossary definition A-25
- System path A-25
- System-supplied job name 6-2
 - Glossary definition A-25
- \$SYSTEM.TDU.TERMINAL_DEFINITIONS 3-11

T

- TABLEND 9-52
- Tape file 11-6
- Tape label attribute
 - Attribute set 11-19
 - Changing 11-12, 16, 26
 - Displaying 11-27
- Tape units 11-1
 - Releasing 11-3
 - Reserving 11-2
- Tape volume, glossary definition A-25
- Tapes 11-1
 - ANSI-labelled 11-5; D-1
 - ANSI file 11-6
 - EOF1 label 11-13; D-19
 - EOF2 label 11-13; D-21
 - EOF3 through EOF9 D-25
 - EOV1 tape label D-22
 - EOV2 tape label D-24
 - Examples 11-28
 - File set 11-6
 - HDR1 label
 - information 11-8; D-8
 - HDR2 label
 - information 11-9; D-15
 - HDR3 through HDR9 D-25
 - Initializing volumes 11-16
 - Label information 11-6
 - Label processing 11-13
 - Multifile volumes 11-12

- NOS/VE HDR2
 - extensions 11-10
 - Reading 11-18
 - Tape file 11-6
 - Tape label attribute 11-19
 - Volume information 11-7
 - Writing 11-17
- Requesting 11-4
- Rewinding 11-5
- Unlabelled 11-5, 32
 - Reading 11-32
 - Skipping tape marks 11-34
 - Writing 11-32
- Target file 6-59
- Task, glossary definition A-25
- \$TASK_NAME function 9-7
- \$TASK_STATUS function 9-7
- TASK/TASKEND 6-34, 35
- Tasks 6-2, 33
 - Active 6-37
 - Asynchronous 6-33
 - Glossary definition A-2
 - Child 6-2
 - Glossary definition A-4
 - Creating 6-34
 - Displaying their status 6-35
 - Executing 6-33
 - Output line 13-46
 - Overview 1-1
 - Parent 6-2
 - Glossary definition A-18
 - Primary 6-37
 - Glossary definition A-19
 - Reading from terminals 13-41
 - Synchronous 6-33
 - Glossary definition A-24
 - Terminating 6-37
 - Utility execution 9-63
 - Writing to terminals 13-46
- Terminal
 - Connection 13-1
 - Definitions 3-10, 11
- Terminal attribute 13-1, 2, 4
 - Applicability 13-17
 - Editing mode 13-21
 - Attribute set 13-5
 - Changing 13-3, 4
 - Defaults 13-17
 - Displaying 13-4
 - Glossary definition A-25
 - Terminal class, glossary
 - definition A-25
 - TERMINAL_CLASS terminal
 - attribute 13-13
 - Terminal definition file, glossary
 - definition A-25
 - Terminal disconnects 3-19
 - Reestablishing 3-19
 - Terminal input 13-41
 - Editing mode 13-42
 - Normal 13-42
 - Transparent 13-44
 - Input buffering 13-41
 - Terminal Interface Program 3-2
 - \$TERMINAL_MODEL
 - function 9-7
 - TERMINAL_MODEL terminal
 - attribute 3-10; 13-14
 - Terminal models 3-10; 13-14
 - Site-defined 3-11
 - TERMINAL_NAME terminal
 - attribute 13-16
 - Terminal output
 - Editing mode 13-46
 - Normal 13-46
 - Transparent 13-49
 - Terminal break 3-16
 - Glossary definition A-25
 - Screen mode 3-16
 - TERMINATE_BREAK_CHARACTER terminal
 - attribute 13-16
 - TERMINATE_COMMAND 3-15
 - TERMINATE_JOB 6-32
 - TERMINATE_OUTPUT 6-52
 - TERMINATE_TASK 6-37
 - TEXT (status field) 2-23; 7-20, 32
 - \$TIME function 9-5
 - \$TRANSLATE function 9-6
 - TRANSPARENT_CHARACTER_MODE connection
 - attribute 13-31
 - TRANSPARENT_FORWARD_CHARACTER connection
 - attribute 13-32
 - TRANSPARENT_LENGTH_MODE connection
 - attribute 13-32

TRANSPARENT_MESSAGE_LENGTH connection attribute Value, glossary definition

TRANSPARENT_MESSAGE_LENGTH connection attribute 13-33
TRANSPARENT_TERMINATE_CHARACTER connection attribute 13-33
TRANSPARENT_TIMEOUT_MODE connection attribute 13-33
\$TRIM function 9-6
Trouble-reporting 12
Typed-ahead input 3-9; 13-45

U

U-type record, glossary definition A-26
Union
Data type 7-36
Definition 7-36
\$UNIQUE function 9-9
Unlabelled tapes 11-5, 32
Reading 11-32
Skipping tape marks 11-34
Writing 11-32
Upper bound (array) 7-6
User-controlled interrupts 3-13
In screen mode 3-16
Pause break 3-15
RESUME_
COMMAND 3-15
TERMINATE_
COMMAND 3-15
Terminate break 3-16
User epilog file 6-9
Changing their names 6-11
USER_FILE_NAME output attribute 6-51
USER_INFORMATION
File attribute 4-54
Job attribute 6-23
Output attribute 6-51
USER_JOB_NAME output attribute 6-51
User name 3-5
Glossary definition A-26
User path, glossary definition A-26

User prolog file 6-9
Changing their names 6-11
User-supplied job names 6-3
User validation 3-21
Utility
Access 9-60
Attributes 9-66
Block 7-2; 8-2, 17
Defining a 9-49
Definition 2-24
Definition file 9-48, 72
Executed as a task 9-63
Execution 2-24
Exiting from 8-14
File 9-48, 60
Formatting 9-47
Functions 2-26
Glossary definition A-26
Interactive input 9-15
Subcommands 2-24
Termination 2-25
\$UTILITY function 9-9
UTILITY/UTILITYEND
Block 9-53, 58
Command 9-51
Command table 9-52
Example 9-49
Execution 9-62
Prompt 9-51
Subcommand information 9-65
Subcommand processors 9-54

V

V-type record, glossary definition A-26
Validation 3-21, 23
\$VALIDATION_LEVEL 3-6
Validation level 3-6
\$VALUE_COUNT function 9-8
Value count, glossary definition A-26
Value, displaying a 7-46
Value element
Definition 2-19
Glossary definition A-27
\$VALUE function 9-8, 32, 44
Value, glossary definition A-26

\$VALUE_KIND function 9-8, 32

Value list

- Definition 2-17
- Glossary definition A-27
- Procedure parameter 9-41
- Range of values 2-18
- Simple values 2-17
- Value sets 2-19

Value set

- Definition 2-19
- Glossary definition A-27

Value set count, glossary definition A-27

VAR OF

- Clause 9-26, 32
- Restrictions 9-32

Variable

- Assigning values 7-7
- Creation 2-30; 7-4, 16
- Data type assignment 2-30
- Data types 7-2
- Default values 7-9
- Definition 2-30; 7-2
- Deletion 7-19
- Display current list 7-19
- Glossary definition A-28
- List 7-19
- Local access 7-10
- Naming conventions 7-3
- Residence 2-30; 7-2, 10
- Scope 7-2, 10, 12

Variable creation

- Examples 7-16
- Explicit 2-30; 7-5, 17
- Implicit 2-30; 7-4, 16
- WHEN/WHENEND**
 - block 8-23

Variable deletion

- Explicit 7-19
- Implicit 7-19

\$VARIABLE function 9-7

Variable name, glossary definition A-28

Variable reference, glossary definition A-28

VERTICAL_PRINT_DENSITY

- Job attribute 6-24
- Output attribute 6-51

VFU load images E-1

VFU_LOAD_PROCEDURE

- Job attribute 6-24
- Output attribute 6-51

\$VNAME function 9-5

Volume accessibility code 11-7

Volume identifier 11-7

VOL1 tape label 11-7; D-5

W

WAIT 8-26; 9-4

Wait queue 6-52

WHEN/WHENEND

- Block 7-2; 8-17, 23
- Exiting from 8-23, 24
- Statement 8-22
- Syntax 8-22
- Variable creation 8-23

WHILE/WHILEND

- Statement 8-7
- Syntax 8-7

Working catalogs 4-7

Glossary definition A-28

Working Environment 1-3

Writing lines to a file 9-12

X

XDCL (Scope of variable) 7-12

XMODEM 14-1

Command 14-9

General 14-6

Special characters 14-8

XREF (Scope of variable) 7-12

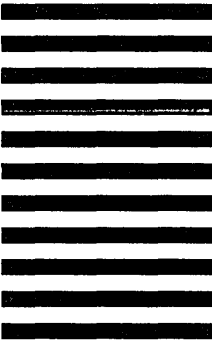
Comments (continued from other side)

fold on dotted line;
seals with tape only.



FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES



BUSINESS REPLY MAIL
First-Class Mail Permit No. 8241 Minneapolis, MN

POSTAGE WILL BE PAID BY ADDRESSEE

CONTROL DATA
Technology & Publications Division
ARH219
4201 N. Lexington Avenue
Arden Hills, MN 55126-9983



We value your comments on this manual. While writing it, we made some assumptions about who would use it and how it would be used. Your comments will help us improve this manual. Please take a few minutes to reply.

Who are you?

- Manager
- Systems analyst or programmer
- Applications programmer
- Operator
- Other _____

How do you use this manual?

- As an overview
- To learn the product or system
- For comprehensive reference
- For quick look-up

What programming languages do you use? _____

How do you like this manual? Check those questions that apply.

- | Yes | Somewhat | No | |
|--------------------------|--------------------------|--------------------------|---------------------------------------------------------------------------------------------------------|
| <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | Is the manual easy to read (print size, page layout, and so on)? |
| <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | Is it easy to understand? |
| <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | Does it tell you what you need to know about the topic? |
| <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | Is the order of topics logical? |
| <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | Are there enough examples? |
| <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | Are the examples helpful? (<input type="checkbox"/> Too simple? <input type="checkbox"/> Too complex?) |
| <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | Is the technical information accurate? |
| <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | Can you easily find what you want? |
| <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | Do the illustrations help you? |

Comments? If applicable, note page and paragraph. Use other side if needed.

Would you like a reply? Yes No

From: _____

Name _____

Company _____

Address _____

Date _____

Phone _____

Please send program listing and output if applicable to your comment.

i

j

