NOS/VE

Commands and Functions

Quick Reference

60464018

CONTROL DATA

NOS/VE

Commands and Functions

Quick Reference

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.

Publication Number 60464018

Manual History

Revision	System Version	PSR Level	Date
A	1.0.2	-	October 1982
B	1.1.1	613	July 1984
C	1.1.3	644	October 1985
D	1.2.1	664	October 1986
E	1.2.2	678	April 1987
F	1.2.3	688	November 1987
G	1.3.1	700	April 1988

Revision G, printed April 1988, documents NOS/VE Version 1.3.1 at PSR level 700. Changes include:

- Commands, subcommands, functions, and control statements are updated.
- The information in the manual is reorganized as follows:
  - Book 1 lists in alphabetical order the commands, functions, and control statements that do not operate under NOS/VE command utilities.
  - Book 2 contains the subcommands and functions that operate under NOS/VE utilities. A separate chapter is devoted to each utility.

This edition obsoletes all previous editions.

Contents

Tables

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 lists the formats of the commands, subcommands, functions, and control statements; briefly describes their parameters; and provides examples of their use.

This manual is part of a set of manuals that describes NOS/VE. For descriptions of other manuals in the set, see The NOS/VE User Manual Set later in this preface.

Audience

This manual is directed to applications programmers. It assumes you are familiar with SCL.

Organization

The information in this manual is divided into two books:

- Book 1 lists in alphabetical order the commands, functions, and control statements that do not operate under NOS/VE utilities. Included are:

APL	IM/Quick
BASIC	LISP
C	MAIL/VE
COBOL	Pascal
CYBIL	Programming environment utilities
File management utility	Prolog
FORTRAN	Screen design facility
IM/DM	Sort/Merge utility

- Book 2 contains the subcommands and functions that operate under NOS/VE utilities. A separate chapter is devoted to each utility (the utilities are listed in the Contents).

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.
Italics	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 NV0 (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

Introduction 1

Introduction 1

This chapter explains how the descriptions of statements are organized and the ways you can specify condition processing.

How to Use the Format Descriptions

The title of each format description lists the statement name and defines the statement as a command, subcommand, function, or control statement. For example:

DELETE_VARIABLE
Command

For a subcommand or function that operates under a utility, the title also includes the abbreviated name of the utility. For example:

POSITION_CURSOR
EDIF Subcommand

Definitions of these terms follow:

- A *command* operates outside a NOS/VE utility.
- A *subcommand* operates under a NOS/VE utility.
- A *control statement* controls the flow of jobs and procedures.
- A *function* returns a value and can operate under a utility or in a command.

The remainder of the description is organized in the following manner:

Purpose Contains a brief description of the purpose of the statement.

Format Contains the format of a statement as follows:

statement name or
plural statement name or
statement name abbreviation
parameter name=value list
parameter name=value list
⋮
parameter name=value list

Plural versions of a name have the same abbreviation as the singular version. The parameter value list is shown in terms of its defined SCL type. Required parameters are shown in boldface type; optional parameters are shown in italic type.

For example:

DELETE_VARIABLE or
DELETE_VARIABLES or
DELV
NAMES=list of name
STATUS=status variable

For a function, the format is shown as follows:

function name
(value,
value,
⋮
value)

For example:

$STRREP
(any,
integer)

The conventions used to depict the formats of statements and functions are for illustration only. The actual entry of a statement or function must follow the SCL rules.

Parameters Contains brief descriptions of each parameter. For commands, alternative spellings and abbreviations of parameter names are also given. For example:

NAMES or **NAME** or **N**

If you do not specify an optional parameter, a default value is used. Every command has a STATUS parameter for error processing (error processing is described later in this chapter).

Remarks Contains restrictions, rules, and additional information.

Examples Contains one or more examples using the statement being described. For example:

```
create_variable i
for i= 1 to 123
  :
forend
delete_variable i
```

Condition Processing

NOS/VE provides the following mechanisms for specifying the action to be taken when an abnormal condition occurs:

- Error processing is available when you include the STATUS parameter on a command and the command terminates with an abnormal status other than a command syntax error.
- Condition handling is available:
  - When you do not include the STATUS parameter on a command and an abnormal condition occurs.
  - When a syntax error occurs in a command.
  - When the job time limit is reached.
  - When a pause break is entered.

Error Processing Using STATUS Parameter

All NOS/VE commands have an optional parameter called STATUS. Including the STATUS parameter on a command causes the SCL interpreter to proceed to the next command when an abnormal condition occurs (with the exception of a command syntax error). Not including the STATUS parameter causes the interpreter to skip succeeding commands in the current block.

To specify a STATUS parameter, you must use a previously declared variable of kind status. This status variable is used by the SCL interpreter to hold the completion status of the command.

By checking the contents of the specified status variable, succeeding commands can alter the flow of statements based on the occurrence of abnormal conditions.

The status variable is a record and contains the following fields:

NORMAL

A boolean value. FALSE indicates the request could not be processed correctly (abnormal status); TRUE indicates the request was processed correctly (normal status).

CONDITION

An 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

A string with a maximum length of 256 characters. This string consists of message parameters that substitute text into the error message associated with the condition (undefined for normal status).

NOTE

The STATUS parameter is listed in each command format but the parameter description is not repeated for each command.

Condition Handling

You can establish condition handlers to be used when errors occur while processing a job. When processing the error, NOS/VE uses the condition handler to determine what action to take:

- For batch jobs, if commands to process the error condition are included in the job, these commands are executed and the job continues. Otherwise, the job is terminated.
- For interactive jobs, the command is terminated. If commands are included to process the error, these commands are executed. Otherwise, you are prompted to enter another command.

The condition handler is called only for errors of severity level ERROR or greater. For more information on severity levels, refer to the NOS/VE Diagnostic Messages manual.

To insert error processing commands, use a WHEN/WHENEND block as follows:

```
when limit_fault do
put_line ' Incrementing time limit by 1 minute.'
change_job_limit name=cp_time ..
  resource_limit=($job_limit(cp_time,accumulator) + 60)
continue retry
whenend
```

Using CONTINUE RETRY as shown in the example tells the system to retry the comand that caused the error. Either omitting CONTINUE or using CONTINUE without RETRY tells the system to resume execution with the command following the erroneous command.

When an error occurs, NOS/VE searches, starting in the current block, for the most recent WHEN block included for that error. The WHEN command specifies which error or errors can be handled by a WHEN block.

The WHEN command can test for the following conditions:

Condition Name	Condition Description
PROGRAM_FAULT	Program terminated in error.
LIMIT_FAULT	Job resource limit was reached.
COMMAND_FAULT	Command had syntax error.
ANY_FAULT	Any of the fault conditions occurred (PROGRAM_FAULT, LIMIT_FAULT, or COMMAND_FAULT).
INTERRUPT	Pause break (terminal interrupt) was entered.

To cancel one or more condition selections made with a WHEN command, use the CANCEL command. For example, to cancel the most recent WHEN block processing for the COMMAND_FAULT and PROGRAM_FAULT conditions, enter:

```
/cancel command_fault program_fault
```

For more information on condition handling, see the NOS/VE System Usage manual.

ACCEPT_LINE Command

Purpose Reads a line from a file into a string variable.

Format

ACCEPT_LINE or
ACCEPT_LINES or
ACCL
VARIABLE=string array
INPUT=file
PROMPT=string
LINE_COUNT=integer variable
STATUS=status variable

Parameters

VARIABLE or **V**

Specifies the variable into which the line is to be read. This variable may be an array of strings. This parameter is required.

INPUT or **I**

Specifies the file from which the line is to be read. This parameter is required.

PROMPT or *P*

Specifies the string to be used for the input prompt. The default prompt is the word SUPPLY, followed by the name of the variable into which the line will be placed. If the file specified by the INPUT parameter is not assigned to a terminal, no prompt is issued.

LINE_COUNT or *LC*

Specifies the SCL integer variable which receives a count of the lines read by the ACCEPT_LINE command.

Remarks

- 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 the 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.

- To use the ACCEPT_LINE command to read successive lines from a file:
  - Attach the file using the ATTACH_FILE command.
  - Specify the $ASIS open position in the file reference for the INPUT parameter of the ACCEPT_LINE command.
  - Use the LINE_COUNT parameter to determine when the end of the file has been reached.
  - Detach the file using the DETACH_FILE command.

  If you do not explicitly attach the file, the $ASIS open position is ignored and the file is accessed from the beginning. See *Examples* below for an example.
- For more information, see the NOS/VE System Usage manual.

Examples

The following example reads successive lines from a file.

```
create_variable n=line_count k=integer
create_variable n=lines k=string d=1..100
attach_file f=$user.some_file
REPEAT
  accept_line v=lines i=$user.some_file.$asis ..
      lc=line_count
  FOR i = 1 TO line_count DO
    "process the line in the variable lines(i)"
  FOREND
UNTIL line_count < $variable(lines, upper_bound)
detach_file f=$user.some_file
```

Using an array variable to read information is more efficient than using a single element variable, since the file is opened fewer times.

$ACCESS_MODE
Function

Purpose Returns a boolean value indicating whether specified access modes are assigned to a file.

Format

$ACCESS_MODE
(file
keyword1
keyword2
keyword3
keyword4
keyword5)

Parameters

file

Specifies the name of the file whose access modes you are querying. This parameter is required.

keyword1

Specifies the access modes you want checked. This parameter is required. Use one or more of the following keywords:

READ

You can read the file.

APPEND

You can append information to the end of the file.

MODIFY

You can alter existing data within the file.

SHORTEN

You can delete data from the end of the file.

WRITE

You can append, modify, and shorten the file.

EXECUTE

You can execute the object code in the file.

ALL

A combination of READ, APPEND, MODIFY, SHORTEN, and EXECUTE access modes.

keyword2

Reserved.

keyword3

Reserved.

keyword4

Reserved.

keyword5

Reserved.

Remarks

- When the file is assigned the access modes you specify, TRUE is returned. When the file is not assigned the access modes you specify, FALSE is returned.
- For further information about functions, see the NOS/VE System Usage manual.

Examples

The following example uses the $ACCESS_MODE function to determine whether a statement list should be executed.

```
if $access_mode(data_file_1,read) then
    .
    ."Statements that need READ access permission."
    .
ifend
```

ADA Command

Purpose

Invokes the compiler and specifies the current sublibrary, the files to be used, and the compiler options to be used.

Format

ADA
INPUT=file
PROGRAM_LIBRARY=file
LIST=file
DEBUG_AIDS=keyword

ERROR=file
ERROR_LEVEL=keyword
LIST_OPTIONS=list of keyword
OPTIMIZATION_LEVEL=keyword
STATUS=status variable

Parameters

INPUT or *I*

Specifies the file that contains the source text to be read. The source input ends when an end-of-partition or an end-of-information is encountered on the source input file. The default value is $INPUT.

PROGRAM_LIBRARY or *PL*

Specifies the name of the current sublibrary. The default is $USER.ADA_PROGRAM_LIBRARY.

LIST or *L*

Specifies the file where the compiler writes the source listing, diagnostics, statistics, and any additional list information specified by the LIST_OPTIONS parameter. The default value is $LIST which, by default, is connected to $NULL.

DEBUG_AIDS or *DA*

Specifies the debug options to be used.

ALL

All of the available options are selected for the DEBUG parameter.

DT

Generates a line number table as part of the object code. This line number table is used by Debug during traceback.

NONE

No debug tables are produced.

If the parameter is omitted, NONE is assumed.

ERROR or *E*

Specifies the file to receive the error listing. The default value is $ERRORS.

ERROR_LEVEL or *EL*

Indicates the minimum severity level of the diagnostics to be listed. The levels, in increasing order of severity, are:

W

WARNING - An error that does not change the meaning of the program or hinder the generation of object code. Also, a construct for which the object code raises a CONSTRAINT_ERROR at run time.

F

FATAL - An illegal construct in the source program has been detected. The compilation continues, but no object code is generated.

C

CATASTROPHIC - An error that causes the compiler to be terminated immediately. No object code is generated.

If the parameter is omitted, W is assumed so all diagnostics are listed.

LIST_OPTIONS or *LO*

Specifies what information appears on the listing file (LIST parameter). Multiple options can be specified; for example, LO=(O,S).

O

Object code listing.

R

Symbolic cross-reference listing of all program entities.

S

Source input listing.

NONE

No list options are selected.

If the parameter is omitted, S is assumed.

OPTIMIZATION_LEVEL or *OL*

Specifies the level of object code optimization.

LOW

Lowest level of production quality code. No optimization is performed.

DEBUG

Generates code to support step mode debugging.

If the DEBUG parameter is omitted, LOW is assumed.

Remarks

- You may want to write a short NOS/VE procedure that invokes the Ada compile command supplying the parameter options you select, then prompts you for your file name. See the Ada for NOS/VE Usage manual for a sample SCL procedure.
- For more information, see the ADA for NOS/VE Usage manual.

Examples

The following compile command specifies program library YOURPL:

```
/ada i=your_file pl=yourpl l=list da=all e=error lo=(r,s)
```

The following compile command uses the default program library:

```
/ada i=your_file l=list da=all e=error lo=(r,s)
```

The following compile command uses all of the default values:

```
/ada i=my_source
```

For Better Performance

Compiler throughput improves when multiple compilation units are submitted. However, if the number of compilation units grows over a certain limit (for example, 50 small compilation units of about 50 lines each) or if the first compilation units are large, a degradation of the throughput actually occurs.

ADA_PROGRAM_LIBRARY_UTILITY Command

Purpose Invokes the Program Library Utility and opens the specified sublibrary.

Format

ADA_PROGRAM_LIBRARY_UTILITY or **ADAPLU**

PROGRAM_LIBRARY=file
INPUT=file
STATUS=status variable

Parameters

PROGRAM_LIBRARY or *PL*

A file name assigned by the user to the current sublibrary at creation time. The default is $USER.ADA_PROGRAM_LIBRARY.

INPUT or *I*

Specifies the file that provides the program library commands. The default value is $INPUT (your terminal in interactive mode).

Remarks For more information, see the ADA for NOS/VE Usage manual.

Examples The following command uses the default parameters:

```
/adaplu
```

The following command invokes PLU commands in batch mode:

```
/adaplu input=adaplu_commands
```

The Program Library Utility is invoked by submitting the following command to the command language interpreter:

```
/ada_program_library_utility
```

or the abbreviation:

```
/adaplu
```

ADMINISTER_RECOVERY_LOG Command

Purpose Begins an Administer_Recovery_Log utility session.

Format

ADMINISTER_RECOVERY_LOG or
ADMRL
STATUS=status variable

Remarks For more information, see the NOS/VE Advanced File Management manual.

Examples The following is the minimal Administer_Recovery_Log session; it does nothing.

```
/administer_recovery_log
admrl/quit
```

To see a list of available subcommands you can type HELP while in this utility.

ADMINISTER_VALIDATIONS Command

Purpose Displays and changes validations.

Format

ADMINISTER_VALIDATIONS or
ADMV
STATUS=status variable

Remarks For more information, see the NOS/VE User Validation manual.

AFTERBURN_OBJECT_TEXT Command

Purpose Expands FORTRAN subroutines and functions in the object text.

Format

AFTERBURN_OBJECT_TEXT or
AFTERBURN_BINARY or
AFTOT or
AFTB
INPUT_OBJECT_TEXT=file
OUTPUT_OBJECT_TEXT=file
LIST=file

EXCLUDE_ENTRY_POINT=list of name
EXCLUDE_FILE_LIST=list of file
FORMAT=keyword
INCLUDE_ENTRY_POINT=list of name
INCLUDE_FILE_LIST=list of file
INCLUDE_MATH_LIBRARY=boolean
LIST_OPTIONS=list of keyword
MAXIMUM_INLINE_SIZE=integer
OPTIMIZATION_LEVEL=keyword
STATUS=status variable

Parameters

INPUT_OBJECT_TEXT or *IOT*

Object file or object library containing the object code you want optimized. If INPUT_OBJECT_TEXT is omitted, $LOCAL.LGO is used.

If you specify an object file, the afterburner binds it into an object module named NEW. The common math library routines are included in NEW if you specify INCLUDE_MATH_LIBRARY=YES. (You can change the name of the module by using the CHANGE_MODULE_ATTRIBUTE subcommand.)

If you specify an object library, it must have been bound with the RETAIN=ALL and INCLUDE_BINARY_SECTION_MAPS=TRUE options set on the CREATE_MODULE subcommand. You should include the common math library in the binding if you want to have math library routines expanded in the object text.

OUTPUT_OBJECT_TEXT or *OOT*

Object file or object library to contain the optimized object code. If OUTPUT_OBJECT_TEXT is omitted, $LOCAL.LGO is used. The OUTPUT_OBJECT_TEXT file overwrites the INPUT_OBJECT_TEXT file if the default file, $LOCAL.LGO, or the same file name, is used for both parameters.

LIST or *L*

File to contain the afterburner output listing. The listing includes information specified by the LIST_OPTIONS parameter. If LIST is omitted, $LIST is used.

EXCLUDE_ENTRY_POINT or *EXCEP*

Specifies the names of entry points you do not want expanded in the object text.

EXCLUDE_FILE_LIST or *EXCFL*

Specifies a file or list of files containing entry point names that you do not want expanded in the object text. Each entry point name in a file should appear on a separate line.

FORMAT or *F*

Specifies the type of the OUTPUT_OBJECT_TEXT parameter. Options are:

LIBRARY or L

Specifies that the OUTPUT_OBJECT_TEXT parameter name is an object library.

FILE or F

Specifies that the OUTPUT_OBJECT_TEXT parameter name is an object file.

SAME

Specifies that the OUTPUT_OBJECT_TEXT parameter is the same type as the INPUT_OBJECT_TEXT parameter. (If the INPUT_OBJECT_TEXT parameter specifies an object library, the OUTPUT_OBJECT_TEXT parameter also specifies an object library. If the INPUT_OBJECT_TEXT parameter specifies an object file, then the OUTPUT_OBJECT_TEXT parameter also specifies an object file.)

If FORMAT is omitted, SAME is used.

INCLUDE_ENTRY_POINT or *INCEP*

Specifies the names of entry points you want expanded in the object text; no other entry points are expanded. If you specify this parameter, the EXCLUDE_FILE_LIST and EXCLUDE_ENTRY_POINT parameters are ignored.

If INCLUDE_ENTRY_POINT and INCLUDE_FILE LIST are omitted, the afterburner considers all modules specified in the INPUT_OBJECT_TEXT parameter for expansion.

INCLUDE_FILE_LIST or *INCFL*

Specifies a file or list of files containing entry point names that you want expanded in the object text; no other entry points are expanded. If you specify this

parameter, the EXCLUDE_FILE_LIST and EXCLUDE_ENTRY_POINT parameters are ignored. Each entry point name in a file should appear on a separate line.

If INCLUDE_ENTRY_POINT and INCLUDE_FILE LIST are omitted, the afterburner considers all modules specified in the INPUT_OBJECT_TEXT parameter for expansion.

INCLUDE_MATH_LIBRARY or *INCML*

Specifies whether you want the common math library routines expanded in the object text. Options are YES, TRUE, or Y to expand the routines and NO, FALSE, or N to not expand them. This parameter is only valid if INPUT_OBJECT_TEXT specifies an object file, such as $LOCAL.LGO.

If INCLUDE_MATH_LIBRARY is omitted, YES is used.

LIST_OPTIONS or *LO*

Specifies the information to appear on the afterburner output listing. Options are:

OBJECT or O

A listing of the optimized object code is provided.

REPORT or R

The afterburner report is generated. The afterburner report indicates which routines were expanded in the object text.

ALL

Selects both the O and R options. The afterburner report appears at the beginning of the object listing.

NONE

No output listing is produced.

If LIST_OPTIONS is omitted, NONE is used.

MAXIMUM_INLINE_SIZE or *MAXIS*

Specifies the maximum size, in bytes, of routines to be expanded in the object text. Routines that are larger than this size are not expanded. If MAXIMUM_INLINE_SIZE is omitted, the maximum size is 2,000 bytes.

OPTIMIZATION_LEVEL or *OL*

Specifies the level of optimization. Options are LOW and HIGH. HIGH selects expansion with additional optimization processing. LOW selects expansion only.

If OPTIMIZATION_LEVEL is omitted, HIGH is used.

Remarks

- Common math library routines are called when a FORTRAN program references an intrinsic function or contains an expression with exponentiation. For example, the expression x**3.1 in a FORTRAN Version 2 program would cause a call to the math library routine MLP$VXTOX. The module names for common math library routines begin with the characters MLP$. The common math library is in the system file $LOCAL.MLF$LIBRARY.

- All debug tables and argument checking information are discarded during the execution of this command.

- The afterburner report states whether a subprogram was expanded in the object text or not and gives the address offset if it was expanded.

- If you specify one or both of the INCLUDE_FILE_LIST or INCLUDE_ENTRY_POINT parameters, remember that these parameters alone specify which routines are considered for expansion. The afterburner may not expand them if the cost/benefit analysis shows that expansion would not be efficient, they are too large, or the routines cause a reference to certain hardware instructions. Routines not listed, including common math library routines, are not considered for expansion.

- The EXCLUDE_FILE_LIST and EXCLUDE_ENTRY_POINT parameters specify which routines you do not want expanded in the object text. Routines that are called once or infrequently, such as error handling routines, are examples of routines which you would not want to expand.

- You can specify both the EXCLUDE_FILE_LIST and EXCLUDE_ENTRY_POINT parameters to indicate which entry points you do not want expanded in the object text. Similarly, you can specify both the INCLUDE_FILE_LIST and INCLUDE_ENTRY_POINT parameters to indicate which entry points you want expanded.

- For more information, see the NOS/VE Object Code Management manual.

ANALYZE_OBJECT_LIBRARY Command

Purpose Begins an ANALYZE_OBJECT_LIBRARY utility session. The subcommands for this object code utility display the internal characteristics of object modules, including: object record counts, section sizes, section attributes, and performance data for modules on an object library or object file.

Format

ANALYZE_OBJECT_LIBRARY or
ANAOL
LIBRARY=file
STATUS=status variable

Parameters *LIBRARY* or *L*

Object library or object file to be analyzed.

If LIBRARY is omitted, you must use the USE_LIBRARY subcommand to specify the object library or object file.

Remarks

- After entering the ANALYZE_OBJECT_LIBRARY command, you can enter any of the ANAOL subcommands. The ANAOL session ends when you enter the QUIT subcommand.

- An object library or file must be specified on the ANALYZE_OBJECT_LIBRARY command or on the USE_LIBRARY subcommand before an ANAOL session can continue.

- For more information, see the NOS/VE Object Code Management manual.

Examples The following is a sequence that enters the ANALYZE_OBJECT_LIBRARY utility, specifies LGO as the file to be analyzed, and displays the characteristics of library LGO.

```
/analyze_object_library lgo
AOL/display_library_analysis
Library Analysis of LGO
Number of modules:  2
Record Analysis

     Identification records:    2
     Libraries:                 2 - items:     10
     Section definitions:       9
     Text records:              21 - items:    519
        .
        .
     Relocation records:        2 - items:      8
     Binding templates:         8
     Transfer symbols:          2
                                ---
       Total records:           84

AOL/quit
```

ANALYZE_PROGRAM_DYNAMICS Command

Purpose Measures program execution characteristics and uses its measurements to restructure the program as a single load module.

Format

ANALYZE_PROGRAM_DYNAMICS or
ANAPD
TARGET_TEXT=file
RESTRUCTURED_MODULE=file
FILE=list of file
PARAMETER=string
LIBRARY=list of file
MODULE=list of any
STARTING_PROCEDURE=any
RESTRUCTURED_MODULE_NAME=any
RESTRUCTURING_COMMANDS=file
STATUS=status variable

Parameters

TARGET_TEXT or TT

File containing the modules to be measured. This parameter is required.

RESTRUCTURED_MODULE or RM

File on which the object library containing the restructured module is written. This parameter is required.

FILE or *FILES* or *F*

Object list for the program. Each module in the specified object files and object libraries is unconditionally included in the program. The list must include the target text file. If FILE is omitted, the object list for the program consists of only the file specified on the TARGET_TEXT parameter.

PARAMETER or *P*

Parameter list passed to the program.

LIBRARY or *LIBRARIES* or *L*

List of object libraries added to the program library list. These object libraries are searched before any libraries in the job library list. The libraries are searched in the order listed.

MODULE or *MODULES* or *M*

Module list. Each module is unconditionally loaded from the object libraries in the program library list.

You use a string value for a module whose name is not an SCL name.

STARTING_PROCEDURE or *SP*

Name of the entry point where execution begins.

You use a string value for an entry point whose name is not an SCL name.

If STARTING_PROCEDURE is omitted, the last transfer symbol encountered during loading is used.

RESTRUCTURED_MODULE_NAME or *RMN*

Name given the restructured module. If RESTRUCTURED_MODULE_NAME is omitted, the module name is the same as the file name.

RESTRUCTURING_COMMANDS or *RC*

File on which the restructuring procedure is written. If RESTRUCTURING_COMMANDS is omitted, the restructuring procedure is discarded when the command completes processing.

Remarks

- To save the procedure used to generate the new load module, you must specify a file on the RESTRUCTURING_COMMANDS parameter.
- ANALYZE_PROGRAM_DYNAMICS does not generate a program profile.
- For more information, see the NOS/VE Object Code Management manual.

Examples

The following command restructures the program on file LGO and writes the new load module NEWLGO on file $USER.NEWLGO. It does not save the restructuring procedure.

```
/analyze_program_dynamics target_text=lgo ..
../restructured_module=$user.newlgo
```

APL Command

Purpose

Activates the APL system.

Format

APL
WORKSPACE=file
TERMINAL_TYPE=keyword
PASSWORD=name
WAIT=boolean
INPUT=file
OUTPUT=file
LIST_OPTIONS=list of keyword
STATUS=status variable

Parameters

WORKSPACE or *WS*

Specifies an APL workspace to be loaded when the APL system starts execution. Default is:

$SYSTEM.APL.CLEARWS

TERMINAL_TYPE or *TT*

Specifies the translation to be performed on the input and output files. Options are APL, COR, UCA, ASCII, and BATCH. Default is APL. Refer to the APL usage manual for descriptions of these options.

PASSWORD or *PW*

Specifies the password required to access the file specified by the WS parameter. Default is no password.

WAIT or *W*

Directs APL to wait for the file specified by the WS parameter if that file is unavailable. If the file is busy and WAIT is OFF, the APL command returns non normal status. Options are ON and OFF. Default is OFF.

INPUT or *I*

Specifies the input file containing APL statements. Default is $INPUT.

OUTPUT or *O*

Specifies the file to receive APL output. Default is $OUTPUT.

LIST_OPTIONS or *LO*

Specifies information to be written to the APL output file. Options are B (suppresses APL banner), S (copies all input to the output file), and P (suppresses prompts in the output file). Default is no options (interactive jobs), or SP (batch jobs).

Remarks

For more information, see the APL Language Definition manual.

Examples The first example calls the APL system for interactive use:

```
APL INPUT=APL_PROG OUTPUT=APL_OUT LIST_OPTIONS=S TT=UCA
```

The following options are specified:

INPUT=APL_PROG

APL statements are read from file APL_PROG.

OUTPUT=APL_OUT

APL output is written to file APL_OUT.

LIST_OPTIONS=S

All input is copied to the output file.

TT=UCA

Terminal type is specified as an ASCII terminal that does not support APL characters (you want lowercase letters to be converted to uppercase on input). For input and output that are not part of the ASCII character set, the $ESCAPE sequences are used.

All other parameters assume default options.

The second example calls the APL system for batch use:

```
APL INPUT=APL_PROG OUTPUT=APL_OUT LIST_OPTIONS=S TT=ASCII
```

The following options are specified:

INPUT=APL_PROG

APL statements are read from file APL_PROG.

OUTPUT=APL_OUT

APL output is written to file APL_OUT.

LIST_OPTIONS=S

All input is copied to the output file.

TT=ASCII

Terminal type is specified as ASCII.

All other parameters assume default options.

ASSEMBLE Command

Remarks Reserved for site personnel, Control Data, or future use.

ATTACH_FILE Command

Purpose Attaches a file to a job.

Format

ATTACH_FILE or
ATTF
FILE=file
LOCAL_FILE_NAME=name
PASSWORD=name or *keyword*
ACCESS_MODES=list of keyword
SHARE_MODES=list of keyword
WAIT=boolean
ERROR_EXIT_PROCEDURE_NAME=name
ERROR_LIMIT=integer
MESSAGE_CONTROL=list of keyword
OPEN_POSITION=keyword
PRIVATE_READ=boolean
STATUS=status variable

Parameters

FILE or **F**

Specifies the file to be attached. Omission of a cycle reference causes $HIGH to be used. This parameter is required.

LOCAL_FILE_NAME or *LFN*

Specifies a local file name (an alias) which can be used by subsequent commands and programs within the job to refer to the file. If this name is already assigned within the job, an error status is returned. Omission causes the permanent file name to be used.

NOTE

Each attach within a job requires a unique LOCAL_FILE_NAME. For this reason, and for compatibility with future NOS/VE releases, it is recommended that you specify a unique LOCAL_FILE_NAME value. Furthermore, it is recommended that you not create an SCL variable with the same name as the LOCAL_FILE_NAME. For example:

```
/create_variable n=lfn1 k=string value=$unique
/create_file f=$user.file local_file_name=$name(lfn1)
```

PASSWORD or *PW*

Specifies the file password. It must match the file password saved with the catalog entry. Otherwise, an error status is returned. Omission or the specification of the keyword NONE causes no password to be used.

ACCESS_MODES or *ACCESS_MODE* or *AM*

Specifies how the file is to be used within the job while attached. Specify the modes you want to use and for which you are validated. Choices are:

READ

You can read the file.

APPEND

You can append information to the end of the file.

MODIFY

You can alter existing data within the file.

EXECUTE

You can execute the object code or SCL procedure in the file.

SHORTEN

You can delete data from the end of the file.

WRITE

You can access the file in SHORTEN, MODIFY, and APPEND modes.

ALL

You can access the file in READ, APPEND, MODIFY, SHORTEN, WRITE, and EXECUTE modes.

If omitted, READ and EXECUTE are assumed.

SHARE_MODES or *SHARE_MODE* or *SM*

Specifies how you will share the file with other jobs while attached to the requesting job. Choices are:

READ

Other jobs may read the file.

APPEND

Other jobs may append data to the end of the file.

MODIFY

Other jobs may alter existing data within the file.

EXECUTE

Other jobs may execute the object code or SCL procedure in the file.

SHORTEN

Other jobs may delete data from the end of the file.

WRITE

Other jobs may attach the file in SHORTEN, MODIFY, and APPEND modes.

ALL

Other jobs may attach the file in READ, APPEND, MODIFY, SHORTEN, WRITE, and EXECUTE modes.

NONE

No other jobs may use the file when it is attached to your job.

Omission causes the share mode to be determined by the value specified on the ACCESS_MODE parameter. If access mode includes APPEND, SHORTEN, or MODIFY, the file is attached with a share mode of NONE. Otherwise, the file is attached with READ and EXECUTE share modes.

WAIT or *W*

This parameter indicates whether the command should wait if the file is temporarily unavailable. Omission causes FALSE to be used (that is, if the file is unavailable, terminate the command and return a status noting the file is busy).

ERROR_EXIT_PROCEDURE_NAME or *EEPN* or *ERROR_EXIT_NAME* or *EEN*

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. Omission causes no error exit procedure to be used.

ERROR_LIMIT or *EL*

Specifies the maximum number of recoverable (nonfatal) file errors that can occur before a fatal error is returned. This parameter is ignored for sequential and byte-addressable files. For details, see the SCL Advanced File Management manual.

MESSAGE_CONTROL or *MC*

Specifies which classes of messages are generated during access of a keyed file. This parameter is ignored for sequential and byte-addressable files. For details, see the SCL Advanced File Management manual.

OPEN_POSITION or *OP*

Specifies the positioning to occur before the file is opened. Options are:

$BOI

Position to beginning-of-information.

$ASIS

Position differs depending on the status of the current instances of open, and whether the requesting task is a private reader of the file. See the NOS/VE System Usage manual for more information.

$EOI

Position to end-of-information.

If an open position is specified on a file reference from a subsequent command, it takes precedence over the open position specified by this command.

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

PRIVATE_READ or *PR*

Specifies whether the attached file is to be accessed for private read. Options are:

TRUE

The attached file is to be privately read. This option has no meaning if any of the following is specified for the ACCESS_MODE parameter:

APPEND
MODIFY
SHORTEN
WRITE
ALL

If none of the previous values are specified on the ACCESS_MODE parameter, this option is the default.

FALSE

Specifies that the attached file is not to be privately read.

For more information on private read, see the NOS/VE System Usage manual.

Remarks

- The local file name can be used for all references to the file by the attaching job.
- Access control information is retrieved from a catalog and is used to validate your request to access the file.
- The access modes and share modes are verified to ensure that the request does not conflict with current file usage.
- The file remains attached until DETACH_FILE is specified.

- Ordinarily, it is not necessary to use this command to reference a file. NOS/VE will automatically attach a file when it is referenced by a command. However, you may need to use this command in the following cases:
  - The file has a password.
  - The subsequent commands require the use of a local file name due to programming language conventions.
  - The file may have multiple cycles and you did not provide a cycle number in the file reference. In this case you may need to use a local file name to ensure that subsequent commands consistently reference the file cycle you attached.
  - The subsequent commands each read part of the file using the $ASIS open position.
  - You need to restrict access to the file by subsequent commands in the job or in other jobs.
- For more information, see the NOS/VE System Usage manual.

Examples

The following example attaches file DATA_FILE_1 from the master catalog, then begins an editing session on the file.

```
/attach_file f=$user.data_file_1 pw=pw_for_data_file_1 ..
../am=(read write) sm=none w=true
/edit_file $user.data_file_1
```

The file is assigned with the access modes READ and WRITE and the share mode NONE. With this access, you are allowed to read and write on the file only. In addition, no one else can access the file while you have it attached.

ATTACH_JOB Command

Purpose Reconnects your terminal to a previously disconnected job.

Format

ATTACH_JOB or
ATTJ
JOB_NAME=name
STATUS=status variable

Parameters *JOB_NAME* or *JN*

Specifies the name of the job to be reconnected. You can omit this parameter if only one job has been disconnected. This parameter is required if more than one job has been disconnected.

Remarks

- You cannot attach a job that originated on a different network from the one you are currently accessing. To attach that job, you must log in again through the network associated with the detached job.
- For more information, see the NOS/VE System Usage manual.

Examples For an example of the ATTACH_JOB command, refer to the example listed under DETACH_JOB.

BACKUP_PERMANENT_FILES Command

Purpose Initiates execution of the utility that backs up permanent files and catalogs. Further processing is directed by utility subcommands.

Format

BACKUP_PERMANENT_FILES or
BACKUP_PERMANENT_FILE or
BACPF
BACKUP_FILE=file
LIST=file
STATUS=status variable

Parameters

BACKUP_FILE or BF

Specifies the file to which backup information is copied. You can specify a file position of beginning-of-information or end-of-information if the file is a mass storage file or a labelled tape. If no file position is specified, or the file is an unlabelled tape, the file is initially positioned to beginning-of-information. This parameter is required.

LIST or *L*

Identifies the file to which a summary of the results of executing the backup utility is written and, optionally, specifies how the file is to be positioned prior to use. Omission causes $LIST to be used.

Remarks

- You can back up only the files for which you have read access.
- For more information, see the NOS/VE System Usage manual.

Examples

The following command initiates a BACKUP_PERMANENT_FILE command utility session. The command specifies that the backed up files are to be written to file BACKED_UP_FILES with the report listing written to file BACKUP_LISTING.

```
/backup_permanent_files bf=backed_up_files ..
../l=backup_listing
```

Following the entry of this command, BACKUP_PERMANENT_FILE subcommands can be entered in response to the following prompt.

```
PUB/
```

BASIC Command

Purpose

Compiles a BASIC source program.

Format

BASIC
INPUT=file
BINARY=file
LIST=file
LIST_OPTIONS=list of keyword
ARRAY_DIMENSIONS=keyword
STATUS=status variable

Parameters *INPUT* or *I*

Specifies the file containing the source program to be compiled. The default input file is $INPUT.

When the BASIC command is invoked using the INPUT parameter, input is expected from the standard file $INPUT. To terminate input and compile what is entered, enter END_OF_INFORMATION after a prompt.

The END_OF_INFORMATION value is a connection attribute. The default value is *EOI. To display the connection attribute value, enter the following SCL command:

```
display_term_conn_default, end_of_information
```

BINARY or *B*

Specifies the file to receive the binary object code. The default binary object file is $LOCAL.LGO.

LIST or *L*

Specifies the file to receive the compiler output listing. The default list file is $LIST.

LIST_OPTIONS or *LO*

Specifies the information that is to appear in the compiler output listing. The list options are:

S

Source listing

O

Object code listing

R

Cross-reference listing

N

None

The default list option is S.

Parameters

BACKUP_FILE or **BF**

Specifies the file to which backup information is copied. You can specify a file position of beginning-of-information or end-of-information if the file is a mass storage file or a labelled tape. If no file position is specified, or the file is an unlabelled tape, the file is initially positioned to beginning-of-information. This parameter is required.

LIST or *L*

Identifies the file to which a summary of the results of executing the backup utility is written and, optionally, specifies how the file is to be positioned prior to use. Omission causes $LIST to be used.

Remarks

- You can back up only the files for which you have read access.
- For more information, see the NOS/VE System Usage manual.

Examples

The following command initiates a BACKUP_PERMANENT_FILE command utility session. The command specifies that the backed up files are to be written to file BACKED_UP_FILES with the report listing written to file BACKUP_LISTING.

```
/backup_permanent_files bf=backed_up_files ..
../l=backup_listing
```

Following the entry of this command, BACKUP_PERMANENT_FILE subcommands can be entered in response to the following prompt.

```
PUB/
```

BASIC Command

Purpose

Compiles a BASIC source program.

Format

BASIC
INPUT=file
BINARY=file
LIST=file
LIST_OPTIONS=list of keyword
ARRAY_DIMENSIONS=keyword
STATUS=status variable

Parameters *INPUT* or *I*

Specifies the file containing the source program to be compiled. The default input file is $INPUT.

When the BASIC command is invoked using the INPUT parameter, input is expected from the standard file $INPUT. To terminate input and compile what is entered, enter END_OF_INFORMATION after a prompt.

The END_OF_INFORMATION value is a connection attribute. The default value is *EOI. To display the connection attribute value, enter the following SCL command:

```
display_term_conn_default, end_of_information
```

BINARY or *B*

Specifies the file to receive the binary object code. The default binary object file is $LOCAL.LGO.

LIST or *L*

Specifies the file to receive the compiler output listing. The default list file is $LIST.

LIST_OPTIONS or *LO*

Specifies the information that is to appear in the compiler output listing. The list options are:

S

Source listing

O

Object code listing

R

Cross-reference listing

N

None

The default list option is S.

ARRAY_DIMENSIONS or *AD*

Indicates whether static or dynamic dimensioning is used for arrays when the program is compiled.

STATIC (S)

Array dimensions are fixed when the program is compiled.

DYNAMIC (D)

Array dimensions can be changed when the program is executed.

The default array is STATIC.

Remarks For more information, see the BASIC for NOS/VE manual.

Examples

```
/BASIC I=BASPROG B=BIN L=COMPLIST LO=(N,O)
```

This command selects the following options:

I=BASPROG

Reads source program from the file BASPROG.

B=BIN

Writes binary code listing to the file BIN.

L=COMPLIST

Writes compiler output listing to the file COMPLIST.

LO=(N,O)

Provides the object code listing but suppresses the source listing.

No status variable is specified.

BLOCK Control Statement

Purpose Groups a sequence of statements into a block.

Format

label: **BLOCK**
statement list
BLOCKEND *label*

Parameters *label*

Specifies the name of the block. The label can be used by EXIT statements within the block.

statement list

Specifies the list of statements that reside in the block.

Remarks

- Exit from the block takes place either through completing execution of the last statement of the statement list or through explicit transfer of control via an EXIT statement.
- For more information, see the NOS/VE System Usage manual.

Examples 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
```

BUILD_REAL_MEMORY Command

Remarks Reserved for site personnel, Control Data, or future use.

C
Command

Purpose Compiles C source text to generate a NOS/VE object module.

NOTE

Before compiling any program written in C, you must first enter:

```
set_working_catalog $user "or another subcatalog"
```

The C command adds libraries to the job command list. These are also required for execution of C binaries. If the C command was not used before executing the program, enter:

```
$system.c.setup
```

Format

C
INPUT=file
BINARY=file
LIST=file
DEBUG_AIDS=keyword
ERROR=file
OPTIMIZATION_LEVEL=keyword
MATH_LIBRARY=keyword
SYMBOL_TABLE_SIZE_MULTIPLIER=integer
STATUS=status variable

Parameters

INPUT or I

File containing the source text to be compiled. The file name must end with the characters _C or _c. (NOS/VE file names are not case-sensitive.)

BINARY or *B*

File to which the object module is written. If the file does not exist, it is created in the working catalog.

If BINARY is omitted, the object module is written on file LGO in the working catalog.

LIST or *L*

File to which error messges are written.

If LIST is omitted, the error messages are written on file LIST in the working catalog.

DEBUG_AIDS or *DA*

Indicates whether the object module is to be written with debug tables so it can be used with Debug.

DT

Generate Debug tables.

NONE

Do not generate Debug tables.

Omission of the parameter causes NONE to be used.

ERROR or *E*

File to which error messages are written.

If ERROR is omitted, the error messages are written on file ERROR in the working catalog.

OPTIMIZATION_LEVEL or *OL* or *OPTIMIZATION* or *OPT*

Indicates whether optimization should be performed.

DEBUG or LOW

No optimization.

HIGH

Peephole optimization.

If OPTIMIZATION_LEVEL is omitted, no optimization (LOW) is performed.

MATH_LIBRARY or *ML*

Indicates which math library to use:

LM

Use the math library which calls the FORTRAN math library (mlf$library).

Parameters

FILE or F

Specifies the name of a NOS/VE temporary file associated with a 170 tape file by a previous CREATE_170_REQUEST command. This parameter is required.

FILE_SET_POSITION or *FSP*

Specifies the position of the 170 tape file to be read.

The FILE_SET_POSITION parameter is not needed for unlabeled tapes because NOS/VE assumes that you wish to read the first file on an unlabeled tape. That is, the value of the FILE_SET_POSITION parameter for a file on an unlabeled 170 tape is BEGINNING_OF_SET, regardless of what you enter.

Only labeled tapes can use all the values of the FILE_SET_POSITION parameter. If you omit the parameter for a labeled tape, the NEXT_FILE position is assumed.

The parameter can have any of the following values:

BEGINNING_OF_SET or BOS

Specifies that the first tape file on the file set is to be read.

CURRENT_FILE or CF

Specifies that the current tape file is to be read. That is, the last tape file accessed will be accessed again. If the tape is positioned at the beginning of the first volume, the first tape file will be read.

FILE_IDENTIFIER_POSITION or FIP

Specifies that the tape file identified by the FILE_IDENTIFIER and GENERATION_NUMBER parameters is to be read.

FILE_SEQUENCE_POSITION or FSP

Specifies that the tape file identified by the FILE_SEQUENCE_NUMBER parameter is to be read.

NEXT_FILE or NF

Specifies that the tape file following the last accessed tape file will be read. If the tape is positioned at the beginning of the first volume, the first tape file will be read.

FILE_IDENTIFIER or *FI*

Specifies a file identifier as a string of 1 to 17 characters. The FILE_IDENTIFIER parameter is for labeled tapes, and it is ignored if you specify it for an unlabeled tape. Each tape file on a multifile set has a unique file identifier. If the FILE_SET_POSITION parameter does not have the FILE_IDENTIFIER_POSITION value, the FILE_IDENTIFIER parameter is ignored.

If you specify the FILE_IDENTIFIER_POSITION value for the FILE_SET_POSITION parameter, the FILE_IDENTIFIER parameter must have a value. If you omit the FILE_IDENTIFIER parameter, a fatal diagnostic is issued.

FILE_SEQUENCE_NUMBER or *FSN*

Specifies the numeric position of a tape file on a multifile set. The position is an unsigned integer in the range 1 through 9999. The FILE_SEQUENCE_NUMBER parameter is for labeled tapes, and it is ignored if you specify it for an unlabeled tape. If the FILE_SET_POSITION parameter is not set to FILE_SEQUENCE_POSITION, the FILE_SEQUENCE_NUMBER parameter value is ignored.

If you specify the FILE_SEQUENCE_POSITION value for the FILE_SET_POSITION parameter, the FILE_SEQUENCE_NUMBER parameter must have a value. If you omit the FILE_SEQUENCE_NUMBER parameter, a fatal diagnostic is issued.

GENERATION_NUMBER or *GN*

Identifies the specific revision of the tape file named by the FILE_IDENTIFIER parameter. The revision is shown as an unsigned integer in the range 1 through 9999. The GENERATION_NUMBER parameter is for labeled tapes, and it is ignored if you specify it for an unlabeled tape. If the FILE_SET_POSITION parameter has the FILE_IDENTIFIER_POSITION value, and the GENERATION_NUMBER parameter is omitted, then the GENERATION_NUMBER parameter value is set to one.

If the FILE_SET_POSITION parameter does not have the FILE_IDENTIFIER_POSITION value, the GENERATION_NUMBER parameter is ignored.

NONE

Math library is not selected.

If MATH_LIBRARY is omitted, no math library is selected.

SYMBOL_TABLE_SIZE_MULTIPLIER or *STSM*

Specifies an integer in the range of 1 through 100. All of the symbol tables are enlarged by this factor.

Omission of this parameter causes 1 to be used.

Remarks For more information, see the C for NOS/VE manual.

CANCEL
Control Statement

Purpose Cancels the most recently selected condition(s).

Format

CANCEL
condition name(s)

Parameters

condition names

Specifies one or more condition selections to be canceled. Multiple condition names are separated by a comma or space. This parameter is required. The following are valid condition names:

PROGRAM_FAULT
LIMIT_FAULT
INTERRUPT
COMMAND_FAULT
ANY_FAULT

Remarks

- SCL ignores any attempt to cancel a nonselected condition.
- For more information, see the NOS/VE System Usage manual.

Examples The following example cancels a previously established LIMIT_FAULT condition:

```
/cancel limit_fault
```

Several conditions can be canceled at the same time, as shown in this example:

```
/cancel command_fault program_fault
```

$CATALOG Function

Purpose Returns the name of the current working catalog.

Format **$CATALOG**

Parameters None.

Remarks For further information about functions, see the NOS/VE System Usage manual.

Examples The following example compares the current working catalog with the catalog $USER. If the working catalog is $USER, it is changed to $LOCAL.

```
if $string($catalog) = $string($fname('$user')) then
    set_working_catalog $local
ifend
```

CHANGE_170_REQUEST Command

Purpose Changes the 170 tape file description in a temporary NOS/VE file formed in a preceding CREATE_170_REQUEST command.

Format **CHANGE_170_REQUEST** or
CHA1R
FILE=file
FILE_SET_POSITION=keyword
FILE_IDENTIFIER=string
FILE_SEQUENCE_NUMBER=integer
GENERATION_NUMBER=integer
INTERNAL_CODE=keyword
CHARACTER_CONVERSION=boolean
BLOCK_TYPE=keyword
RECORD_TYPE=keyword
MAXIMUM_BLOCK_LENGTH=integer
MAXIMUM_RECORD_LENGTH=integer
TAPE_FORMAT=keyword
STATUS=status variable

INTERNAL_CODE or *IC*

Specifies the character set of the data on the tape volume. If you omit the parameter, its value is left at its previous setting. The INTERNAL_CODE parameter can have the following values:

AS6

6/12-bit ASCII

AS8

8/12-bit ASCII

D63

63-character display code

D64

64-character display code

CHARACTER_CONVERSION or *CC*

Boolean value specifies whether or not file data is to be converted to or from the character set specified by the INTERNAL_CODE parameter. If you omit the CHARACTER_CONVERSION parameter, its value is left at its previous setting.

Of the tape file migration methods, only FMA and FMU automatically do character conversion in addition to any conversion specified by the CHARACTER_CONVERSION parameter value.

To obtain a properly migrated tape file, you usually want to set the CHARACTER_CONVERSION parameter to FALSE if you use FMA or FMU to migrate. Otherwise you convert your tape file data twice. Set the parameter to TRUE if you use any other tape file migration method. Otherwise, you do not convert your tape file data at all.

BLOCK_TYPE or *BT*

Specifies the block type of the 170 input tape file. If you omit this parameter, its value is left at its previous setting. Its value can be either of the following:

INTERNAL or I

Internal blocking

CHARACTER_COUNT or CC

Character count blocking

RECORD_TYPE or *RT*

Specifies the record type of the 170 tape file. If you omit this parameter, its value is left at its previous setting. Its value can be any of the following:

CONTROL_WORD, CW, or W

Control word

FIXED_LENGTH, FL, or F

Fixed length

SYSTEM_RECORD, SR, or S

System record

ZERO_BYTE, ZB, or Z

Zero-byte

MAXIMUM_BLOCK_LENGTH or *MAXBL* or *MBL*

Specifies with an unsigned integer the maximum length in 6-bit bytes of a block in the 170 tape file. The system maximum for this parameter is 2,147,483,615. If you omit this parameter, its value is left at its previous setting.

MAXIMUM_RECORD_LENGTH or *MAXRL* or *MRL*

Specifies with an unsigned integer the maximum length in 6-bit bytes of a record in the 170 tape file. The system maximum for this parameter is 4,398,046,511,103. If you omit this parameter, its value is left at its previous setting.

TAPE_FORMAT or *TF*

Specifies the tape format of the 170 tape file. If you omit this parameter, its value is left at its previous setting. The possible values for this parameter are:

NOS_INTERNAL, NI, or I

Internal, NOS default tape format

NOS_BE_INTERNAL, NBI, or SI

SCOPE internal and NOS/BE default tape format

STRANGER or S

Stranger

LONG_STRANGER, LS, or L

Long block stranger

Remarks

- This command is handy when you wish to use the same temporary NOS/VE file to reference several tape files from the same multifile set on tape. You simply use the CHANGE_170_REQUEST command to change the tape file description.

- In general, when you omit a parameter from the CHANGE_170_REQUEST command, the value of that parameter is not changed. Thus, usually, you need to include only those parameters whose settings change for the new 170 tape file.

  The two exceptions to this rule are the FILE_SET_POSITION and the GENERATION_NUMBER parameters. When you omit these parameters, their values become the default values of the parameters. For the FILE_SET_POSITION parameter, the default is BEGINNING_OF_SET for an unlabeled tape and NEXT_FILE for a labeled tape. The default value for the GENERATION_NUMBER parameter is 1.

Examples

Suppose that you have associated the first tape file of a multifile set on a 170 labeled tape with the temporary NOS/VE file, MULTI_FILES. Once you have processed this first tape file, you wish to access the fifth tape file on the same multifile set. The following command associates the fifth tape file with MULTI_FILES.

```
/change_170_request  file=multi_files ..
../file_set_position=file_sequence_position ..
../file_sequence_number=5 ..
../internal_code=d63 ..
../character_conversion=true ..
../block_type=character_count ..
../record_type=system_record ..
../maximum_block_length=6000 ..
../maximum_record_length=300 ..
../tape_format=nos_be_internal
```

The tape file has the following characteristics:

- Found from its position within the multifile set.
- Data to be converted from its 63 character display code.
- Character count blocking and system record type.
- Maximum length of a block is 6000 6-bit bytes; of a record 300 6-bit bytes.
- NOS/BE default tape format.

CHANGE_7600_REQUEST Command

Purpose Changes the 7600 tape file description in a temporary NOS/VE file formed in a preceding CREATE_7600_REQUEST command.

Format

CHANGE_7600_REQUEST or
CHA7R
FILE=file
FILE_SET_POSITION=keyword
FILE_IDENTIFIER=string
FILE_SEQUENCE_NUMBER=integer
GENERATION_NUMBER=integer
INTERNAL_CODE=keyword
CHARACTER_CONVERSION=boolean
BLOCK_TYPE=keyword
RECORD_TYPE=keyword
MAXIMUM_BLOCK_LENGTH=integer
MAXIMUM_RECORD_LENGTH=integer
STATUS=status variable

Parameters

FILE or **F**

Specifies the name of a NOS/VE temporary file associated with a 7600 tape file by a previous CREATE_7600_REQUEST command. This parameter is required.

FILE_SET_POSITION or *FSP*

Specifies the position of the 7600 tape file to be read.

The FILE_SET_POSITION parameter is not needed for unlabeled tapes because NOS/VE assumes that you wish to read the first file on an unlabeled tape. That is, the

value of the FILE_SET_POSITION parameter for a file on an unlabeled 7600 tape is BEGINNING_OF_SET, regardless of what you enter.

Only labeled tapes can use all the values of the FILE_SET_POSITION parameter. If you omit the parameter for a labeled tape, the NEXT_FILE position is assumed.

The parameter can have any of the following values:

BEGINNING_OF_SET or BOS

Specifies that the first tape file on the file set is to be read.

CURRENT_FILE or CF

Specifies that the current tape file is to be read. That is, the last tape file accessed will be accessed again. If the tape is positioned at the beginning of the first volume, the first tape file will be read.

FILE_IDENTIFIER_POSITION or FIP

Specifies that the tape file identified by the FILE_IDENTIFIER and GENERATION_NUMBER parameters is to be read.

FILE_SEQUENCE_POSITION or FSP

Specifies that the tape file identified by the FILE_SEQUENCE_NUMBER parameter is to be read.

NEXT_FILE or NF

Specifies that the tape file following the last accessed tape file will be read. If the tape is positioned at the beginning of the first volume, the first tape file will be read.

FILE_IDENTIFIER or *FI*

Specifies a file identifier as a string of 1 to 17 characters. The FILE_IDENTIFIER parameter is for labeled tapes, and it is ignored if you specify it for an unlabeled tape. Each tape file on a multifile set has a unique file identifier. If the FILE_SET_POSITION parameter does not have the FILE_IDENTIFIER_POSITION value, the FILE_IDENTIFIER parameter is ignored.

If you specify the FILE_IDENTIFIER_POSITION value for the FILE_SET_POSITION parameter, the FILE_IDENTIFIER parameter must have a value. If you omit the FILE_IDENTIFIER parameter, a fatal diagnostic is issued.

FILE_SEQUENCE_NUMBER or *FSN*

Specifies the numeric position of a tape file on a multifile set. The position is an unsigned integer in the range 1 through 9999. The FILE_SEQUENCE_NUMBER parameter is for labeled tapes, and it is ignored if you specify it for an unlabeled tape. If the FILE_SET_POSITION parameter is not set to FILE_SEQUENCE_POSITION, the FILE_SEQUENCE_NUMBER parameter value is ignored.

If you specify the FILE_SEQUENCE_POSITION value for the FILE_SET_POSITION parameter, the FILE_SEQUENCE_NUMBER parameter must have a value. If you omit the FILE_SEQUENCE_NUMBER parameter, a fatal diagnostic is issued.

GENERATION_NUMBER or *GN*

Identifies the specific revision of the tape file named by the FILE_IDENTIFIER parameter. The revision is shown as an unsigned integer in the range 1 through 9999. The GENERATION_NUMBER parameter is for labeled tapes, and it is ignored if you specify it for an unlabeled tape. If the FILE_SET_POSITION parameter has the FILE_IDENTIFIER_POSITION value, and the GENERATION_NUMBER parameter is omitted, then the GENERATION_NUMBER parameter value is set to one.

If the FILE_SET_POSITION parameter does not have the FILE_IDENTIFIER_POSITION value, the GENERATION_NUMBER parameter is ignored.

INTERNAL_CODE or *IC*

Specifies the character set of the data on the tape volume. If you omit the parameter, its value is left at its previous setting. The INTERNAL_CODE parameter can have the following values:

AS6

6/12-bit ASCII

AS8

8/12-bit ASCII

D63

63-character display code

D64

64-character display code

CHARACTER_CONVERSION or *CC*

Boolean value specifies whether or not file data is to be converted to or from the character set specified by the INTERNAL_CODE parameter. If you omit the CHARACTER_CONVERSION parameter, its value is left at its previous setting.

Of the tape file migration methods, only FMA and FMU automatically do character conversion in addition to any conversion specified by the CHARACTER_CONVERSION parameter value.

To obtain a properly migrated tape file, you usually want to set the CHARACTER_CONVERSION parameter to FALSE if you use FMA or FMU to migrate. Otherwise you convert your tape file data twice. Set the parameter to TRUE if you use any other tape file migration method. Otherwise, you do not convert your tape file data at all.

BLOCK_TYPE or *BT*

Specifies the block type of the 7600 input tape file. If you omit this parameter, its value is left at its previous setting. Its value can be either of the following:

INTERNAL or I

Internal blocking

CHARACTER_COUNT or CC

Character count blocking

RECORD_TYPE or *RT*

Specifies the record type of the 7600 tape file. If you omit this parameter, its value is left at its previous setting. Its value can be any of the following:

CONTROL_WORD, CW, or W
Control word

FIXED_LENGTH, FL, or F
Fixed length

SYSTEM_RECORD, SR, or S
System record

ZERO_BYTE, ZB, or Z
Zero-byte

MAXIMUM_BLOCK_LENGTH or *MAXBL* or *MBL*

Specifies with an unsigned integer the maximum length in 6-bit bytes of a block in the 7600 tape file. The system maximum for this parameter is 2,147,483,615. If you omit this parameter, its value is left at its previous setting.

MAXIMUM_RECORD_LENGTH or *MAXRL* or *MRL*

Specifies with an unsigned integer the maximum length in 6-bit bytes of a record in the 7600 tape file. The system maximum for this parameter is 4,398,046,511,103. If you omit this parameter, its value is left at its previous setting.

Remarks

- This command is handy when you wish to use the same temporary NOS/VE file to reference several tape files from the same multifile set on tape. You simply use the CHANGE_7600_REQUEST command to change the tape file description.

- In general, when you omit a parameter from the CHANGE_7600_REQUEST command, the value of that parameter is not changed. Thus, usually, you need to include only those parameters whose settings change for the new 7600 tape file.

  The two exceptions to this rule are the FILE_SET_POSITION and the GENERATION_NUMBER parameters. When you omit these parameters, their values become the default values of the parameters. For the FILE_SET_POSITION parameter, the default is BEGINNING_OF_SET for an unlabeled tape and NEXT_FILE for a labeled tape. The default value for the GENERATION_NUMBER parameter is 1.

Examples Suppose that you have associated a tape file on an unlabelled 7600 tape with the temporary NOS/VE file, TAPE_7600. Once you have processed this tape file, you wish to access it again. The file is at the beginning of the volume. The following command fully describes the tape file and again associates the tape file with TAPE_7600:

```
/change_7600_request  file=tape_7600 ..
../file_set_position=beginning_of_set ..
../internal_code=as8 ..
../character_conversion=true ..
../block_type=character_count ..
../record_type=fixed_length ..
../maximum_block_length=8000 ..
../maximum_record_length=250
```

The tape file has the following characteristics:

- Data to be converted from its 8/12-bit ASCII character set.
- Character count blocking and fixed length record type.
- Maximum length of a block is 8000 6-bit bytes; of a record 250 6-bit bytes.

CHANGE_ALTERNATE_INDEXES Command

Purpose Initiates execution of the CREATE_ALTERNATE_INDEXES command utility. The utility can create, delete, and display alternate-key definitions in a keyed file.

Format

CHANGE_ALTERNATE_INDEXES or
CHANGE_ALTERNATE_INDEX or
CHANGE_ALTERNATE_INDICES or
CHAAI
INPUT=list of any
STATUS=status variable

Parameters

INPUT or I

Keyed file to be processed by the utility. The file permissions required depend on the subcommands entered during the utility as described in the Remarks. This parameter is required.

To specify a nested file, first specify the file reference and then the nested-file name, enclosed in parentheses.

Remarks

- The command utility prompt is:

  creai/

- In response to the creai/ prompt, you can enter certain SCL commands and any of these subcommands:

  QUIT

  DISPLAY_KEY_DEFINITIONS

  CREATE_KEY_DEFINITIONS

  DELETE_KEY_DEFINITIONS

  CANCEL_KEY_DEFINITIONS

  APPLY_KEY_DEFINITIONS

- The CREATE_ALTERNATE_INDEXES utility creates a keyed file under these conditions:

  - The file set does not exist and,
  - A SET_FILE_ATTRIBUTES command for the file was entered before the CREATE_ALTERNATE_INDEXES command.

  The SET_FILE_ATTRIBUTES command is required because the default file-organization attribute is sequential and CHANGE_ALTERNATE_INDEXES only processes keyed files.

- The CHANGE_ALTERNATE_INDEXES command does not check your file permissions. The subcommands you enter in the utility session check that you have the required permissions to do the operation.

- To display key definitions, you must have at least read permission.

- To create, delete, cancel, or apply key definitions you must have at least the three permissions: append, modify, and shorten.

- For more information, see the NOS/VE Advanced File Management Usage manual.

Examples This command begins a utility session that displays the alternate key definitions of keyed file $USER.IS_FILE.

```
/change_alternate_indexes input=$user.is_file
creai/display_key_definitions key_names=all display_options=brief
                Display_Key_Definitions        NOS/VE Keyed File Utilities 1.1
File = :NVE.USER99.IS_FILE

 KEY NAME                  POSITION    LENGTH   TYPE         STATE
 ------------------------ --------    ------   ----------   --------------
 ALTERNATE_KEY_1                0        10   uncollated   Exists in file
creai/quit      "The APPLY_KEY_DEFINITIONS parameter is not required here"
                "because no creation or deletion requests are pending."
```

CHANGE_BACKUP_LABEL_TYPE Command

Purpose Changes the job default label type for permanent file backup and restore operations which have a backup file assigned to a tape device. It does not affect the default label type for other types of tape file usage.

Format **CHANGE_BACKUP_LABEL_TYPE** or **CHABLT**

FILE_LABEL_TYPE=keyword

STATUS=status variable

Parameters **FILE_LABEL_TYPE or FLT**

Specifies the tape label type. You must select:

UNLABELLED (U) for unlabelled tapes.

or

LABELLED (L) for labelled tapes.

Remarks For more information, see the NOS/VE System Usage manual.

Examples The following example enables you to use unlabelled tapes for permanent file backup:

```
/change_backup_label_type..
../file_label_type = unlabelled
```

CHANGE_CATALOG_CONTENTS Command

Purpose Deletes damage conditions from files in the specified catalogs. This command can also delete catalog entries for files for which no file cycle data exists in mass storage.

Format

CHANGE_CATALOG_CONTENTS or
CHANGE_CATALOG_CONTENT or
CHACC
CATALOG=file or keyword
DELETE_DAMAGE_CONDITION=list of keyword
DELETE_UNRECONCILED_FILES=boolean
STATUS=status variable

Parameters

CATALOG or C

Specifies the catalog for which to change the contents. The keyword ALL specifies all catalogs for all families in the system.

DELETE_DAMAGE_CONDITION or *DELETE_DAMAGE_CONDITIONS* or *DDC*

Specifies the damage condition to delete from the files in the specified catalogs. DELETE_DAMAGE_CONDITION has the following values:

PARENT_CATALOG_RESTORED or PCR

This damage condition warns users when catalogs have been restored.

RESPF_MODIFICATION_MISMATCH or RMM

The damage condition warns users when a file restored from a backup tape has a different modification date than what is recorded in the file's catalog.

If this parameter is omitted, RESPF_MODIFICATION_MISMATCH is used.

DELETE_UNRECONCILED_FILES or *DELETE_LOST_CYCLES* or *DLC* or *DUF*

Specifies whether to delete catalog entries for all files for which no file cycle data exists in mass storage. DELETE_UNRECONCILED_FILES has the following values:

TRUE

Delete unreconciled files.

FALSE

Do not delete unreconciled files.

Remarks For more information, see the NOS/VE System Performance and Maintenance manual, Volume 2.

Examples This example deletes the RESPF_MODIFICATION_MISMATCH damage condition from all catalogs in all families:

```
/change_catalog_contents ..
/delete_damage_condition=respf_modification_mismatch
```

CHANGE_CATALOG_ENTRY Command

Purpose Changes the file name, cycle, password, log selection, retention period, charge identification, and damage condition associated with a file.

Format

CHANGE_CATALOG_ENTRY or
CHACE
FILE=file
PASSWORD=name or *keyword*
NEW_FILE_NAME=name
NEW_CYCLE=integer
NEW_PASSWORD=name or *keyword*
NEW_LOG=boolean
NEW_RETENTION=integer
NEW_ACCOUNT_PROJECT=boolean
DELETE_DAMAGE_CONDITION=list of keyword
STATUS=status variable

Parameters

FILE or **F**

Specifies the file whose information is to be changed. Omission of a cycle causes $HIGH to be used if it is applicable to the command (that is, changing the cycle number, retention, or damage condition). Otherwise, the cycle is ignored. This parameter is required.

PASSWORD or *PW*

Specifies the current file password. It must match the file password in the catalog entry or an abnormal status is returned. The keyword NONE indicates that no password has been specified for the file. Omission causes NONE to be used.

NEW_FILE_NAME or *NFN*

Specifies the new permanent file name to be associated with the file. All existing cycles of the file contain this new name. If the new name already exists in the catalog, an abnormal status is returned. Omission causes the current name to be retained.

NEW_CYCLE or *NC*

Specifies the new cycle number (from 1 through 999) to be associated with a file cycle. If the specific number already exists, an abnormal status is returned. Omission causes the current cycle number to be retained.

NEW_PASSWORD or *NPW*

Specifies the new password to be associated with all cycles of the file. The keyword NONE indicates that no password is to be associated with the file. Omission causes the current password to be retained.

NEW_LOG or *NL*

Specifies whether a new file access logging option is to be associated with all cycles of the file. Omission causes the current log option to be retained.

NEW_RETENTION or *NR*

Specifies a new retention period for the specified file cycle. It determines the number of days (from 1 through 999) from the current date that a file cycle is to be retained. Omission causes the current retention period to be retained.

NEW_ACCOUNT_PROJECT or *NAP*

Specifies whether new account and project identifiers are to be established for the file. These identifications apply to all cycles of the file. The values are determined by NOS/VE according to the project and account identification

associated with the requesting job. Omission causes the current account and project identifiers associated with the file to be retained.

DELETE_DAMAGE_CONDITION or *DDC*

Specifies the cycle damage condition to be deleted from the file cycle's catalog registration. Keyword:

RESPF_MODIFICATION_MISMATCH (RMM)

Indicates that a file cycle was restored even though the last modification date and time were different from the backed-up permanent file.

Omission, if a cycle damage condition exists, causes a diagnostic message to be displayed.

Remarks

- This request can be issued only if you have CONTROL permission. If the file has a password associated with it, you must also specify the password.
- A retention period of 999 indicates an infinite retention period.
- For more information, see the NOS/VE System Usage manual.

Examples

The following example changes the name, password, log option, and retention period for cycle number 1 of file DATA_FILE_1 in subcatalog CATALOG_1 in the master catalog.

```
/change_catalog_entry $user.catalog_1.data_file_1.1 ..
../nfn=data_file_0 nc=87 npw=new_data_0_pw nl=true nr=60
/disce $user.catalog_1.data_file_0
    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: 1987-03-19 10:32:28.750,
    LAST ACCESS DATE AND TIME: 1987-03-19 10:32:46.252,
    LAST MODIFICATION DATE AND TIME: 1987-03-19 10:32:46.252,
    EXPIRATION DATE: 1987-05-18
    CYCLE NUMBER:      2, ACCESS COUNT: 2,
    CREATION DATE AND TIME: 1987-03-19 10:32:28.923,
    LAST ACCESS DATE AND TIME: 1987-03-19 10:32:46.252,
    LAST MODIFICATION DATE AND TIME: 1987-03-19 10:32:46.252,
    EXPIRATION DATE: NONE
/disc $user.catalog_1 file
 DATA_FILE_0
    NUMBER OF CYCLES:  2, ACCOUNT: D5927, PROJECT: P693N354
 DATA_FILE_2
    NUMBER OF CYCLES:  1, ACCOUNT: D5927, PROJECT: P693N354
```

The name of the file is changed from DATA_FILE_1 to DATA_FILE_0 for all cycles in the file, the cycle number for cycle 1 is changed to 87 (other cycle numbers remain unchanged), the password for the file is changed from null to NEW_DATA_0_PW, the log activity selection for the file is changed to TRUE, and the retention period for cycle 87 is changed to 60 days.

CHANGE_COMMAND_SEARCH_MODE Command

Purpose Changes the command list search mode.

Format

CHANGE_COMMAND_SEARCH_MODE or **CHACSM**
SEARCH_MODE=keyword
STATUS=status variable

Parameters

SEARCH_MODE or SM

Specifies the new search mode to be associated with the current command list. Choose one of the following keywords:

GLOBAL (G)

All entries in the command list can be searched. Commands specified by path name and command name can be executed.

RESTRICTED (R)

All entries in the command list can be searched. However, for a search to proceed beyond the first entry in the command list, the command you enter must be preceded by a slash (/). Commands specified by path name and command name can be executed.

EXCLUSIVE (E)

Only the entry at the beginning of the command list is searched for a command. Commands that are specified by path and command name are not allowed.

This parameter is required.

Remarks

- This command may not be used when the command list search mode is EXCLUSIVE, or if a command utility is active and the command list search mode is RESTRICTED.
- For more information, see the NOS/VE System Usage manual.

CHANGE_CONNECTION_ATTRIBUTES Command

Purpose Changes the terminal file's connection attributes.

Format

CHANGE_CONNECTION_ATTRIBUTES or
CHANGE_CONNECTION_ATTRIBUTE or
CHANGE_TERM_CONN_ATTRIBUTE or
CHANGE_TERM_CONN_ATTRIBUTES or
CHATCA or
CHACA
TERMINAL_FILE_NAME=file
ATTENTION_CHARACTER_ACTION=integer
BREAK_KEY_ACTION=integer
END_OF_INFORMATION=string
INPUT_BLOCK_SIZE=integer
INPUT_EDITING_MODE=keyword
INPUT_OUTPUT_MODE=keyword
INPUT_TIMEOUT=boolean
INPUT_TIMEOUT_LENGTH=integer
INPUT_TIMEOUT_PURGE=boolean
PARTIAL_CHARACTER_FORWARDING=boolean
PROMPT_FILE=file
PROMPT_STRING=string
STORE_BACKSPACE_CHARACTER=boolean
STORE_NULS_DELS=boolean
TRANSPARENT_CHARACTER_MODE=keyword
TRANSPARENT_FORWARD_CHARACTER=list of string
TRANSPARENT_LENGTH_MODE=keyword
TRANSPARENT_MESSAGE_LENGTH=integer
TRANSPARENT_TERMINATE_CHARACTER=list of string
TRANSPARENT_TIMEOUT_MODE=keyword
STATUS=status variable

Parameters

TERMINAL_FILE_NAME or TFN

Specifies the terminal file name. This parameter is required.

ATTENTION_CHARACTER_ACTION or ACA

Specifies the type of user interrupt command simulated by the network when the character defined by the ATTENTION_CHARACTER attribute is received from the terminal. If the ATTENTION_CHARACTER is set to NUL, this attribute has no effect. Values are:

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 or BKA

Specifies the type of user interrupt command simulated by the network when the break key is pressed at the terminal. Values are:

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 or 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.

INPUT_BLOCK_SIZE or *IBS*

Specifies the maximum number of characters (80 to 2,000) 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 or *IEM*

Specifies how the network edits the data received from the terminal. This attribute also determines how the network edits ouput sent to the terminal.

NORMAL (N)

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 (T)

TRANSPARENT editing mode is in effect. The network forwards input to NOS/VE without converting or deleting any special codes or characters.

INPUT_OUTPUT_MODE or *IOM*

Specifies how the network coordinates the terminal input and output streams. Values are:

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.

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.

FULL_DUPLEX (FD)

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.

INPUT_TIMEOUT or *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. Values are:

TRUE

NOS/VE limits the task wait time.

FALSE

NOS/VE does not limit the task wait time.

INPUT_TIMEOUT_LENGTH or *ITL*

Specifies the maximum number of milliseconds (0 to 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 an abnormal status.

INPUT_TIMEOUT_PURGE or *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 (zero).

PARTIAL_CHARACTER_FORWARDING or *PCF*

Specifies whether the network forwards a partial input line when an END_PARTIAL_CHARACTER is received from the terminal. Values are:

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 or *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.

PROMPT_STRING or *PS*

Specifies the string written to the prompt file when a task reads from the terminal file.

STORE_BACKSPACE_CHARACTER or *SBC*

Specifies how the network processes the character specified as the BACKSPACE_CHARACTER when it is received from the terminal. Values are:

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_NULS_DELS or *SND*

Specifies whether the network stores or discards the NUL and DEL characters when they are received from the terminal. Values are:

TRUE

The network stores the NUL and 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 or *TCM*

Specifies the action the network takes when a TRANSPARENT_FORWARD_CHARACTER or a TRANSPARENT_TERMINATE_CHARACTER is received from the terminal. Values are:

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.

FWD_TERMINATE (FD)

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 or *TFC*

Specifies a list of 1- to 4-character strings; any one of these strings is recognized by the network as the TRANSPARENT mode forwarding character.

TRANSPARENT_LENGTH_MODE or *TLM*

Specifies the action the network takes when it receives the number of characters specified by the TRANSPARENT_MESSAGE_LENGTH connection attribute. Values are:

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 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 or *TML*

Specifies the minimum number of characters (1 to 32,767) forwarded in each transparent input message.

TRANSPARENT_TERMINATE_CHARACTER or *TTC*

Specifies a list of 1- to 4-character strings; any one of these strings is recognized by the network as the TRANSPARENT mode terminating character.

TRANSPARENT_TIMEOUT_MODE or *TTM*

Specifies the action the network takes when no input is received from the terminal for an interval of 400 milliseconds or more. Values are:

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.

Remarks For more information, see the NOS/VE System Usage manual.

CHANGE_FILE_ATTRIBUTES Command

Purpose Changes the file attributes for an existing file.

Format

CHANGE_FILE_ATTRIBUTES or
CHANGE_FILE_ATTRIBUTE or
CHAFA
FILE=file
FILE_ACCESS_PROCEDURE_NAME=name
FILE_CONTENTS=name
FILE_LIMIT=integer
FILE_PROCESSOR=name
FILE_STRUCTURE=name
FORCED_WRITE=boolean or *keyword*
LINE_NUMBER=list of integer
LOADING_FACTOR=integer
LOCK_EXPIRATION_TIME=integer
LOGGING_OPTIONS=list of keyword
LOG_RESIDENCE=file or *keyword*
RECORD_LIMIT=integer
RING_ATTRIBUTES=list of integer
STATEMENT_IDENTIFIER=list of integer
USER_INFORMATION=string
STATUS=status variable

Parameters

FILE or **F**

Specifies the file for which attributes are altered. This parameter is required.

FILE_ACCESS_PROCEDURE_NAME or *FAPN* or *FILE_ACCESS_PROCEDURE* or *FAP*

Specifies the name of a CYBIL procedure that intervenes in the calling sequence between users of the file and the access method. Omission causes the current file access procedure to be retained.

The keyword NONE can also be used. For details, see the CYBIL File Management manual.

FILE_CONTENTS or *FILE_CONTENT* or *FC*

Specifies the type of data contained in the file. Omission causes the file content to be retained. The FILE_CONTENT value for a keyed file must not be changed to LIST.

FILE_LIMIT or *FL*

Specifies a new limit for the maximum length of the file. This new value cannot be less than the old file limit. Omission causes the current maximum length to be retained.

FILE_PROCESSOR or *FP*

Specifies the name of the processor of the file. Omission causes the current file processor to be retained.

FILE_STRUCTURE or *FS*

Specifies the structure of the file. Omission causes the current file structure to be retained.

FORCED_WRITE or *FW*

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. Omission causes the current entry to be retained.

LINE_NUMBER or *LN*

Specifies the length and location of a line number in each record of a file. Omission causes the current line number to be retained.

LOADING_FACTOR or *LF*

Reserved for site personnel, Control Data, or future use.

LOCK_EXPIRATION_TIME or *LET*

Specifies the number of milliseconds between the time a lock is granted and the time that it could expire. Values can be any integer from 0 to 604,800,000 (1 week). If the value is changed to 0, locks for the file do not expire.

LOGGING_OPTIONS or *LOGGING_OPTION* or *LO*

Enables the use of keyed-file recovery options. Options are:

ALL

All logging options are enabled.

ENABLE_PARCELS (EP)

Reserved for future use.

ENABLE_MEDIA_RECOVERY (EMR)

An update recovery log is to be maintained for the keyed-file.

ENABLE_REQUEST_RECOVERY (ERR)

An automatic close upon task abort removes from the keyed-file any partially completed update operation caused by system failure.

NONE

No logging options are enabled. This is the default.

For more information on logging options, see the SCL Advanced File Management Usage manual, or the CYBIL Keyed-File and Sort/Merge Interface Usage manual.

LOG_RESIDENCE or *LR*

Specifies the catalog path for the keyed-file's update recovery log. The log must be created by the ADMINISTER_RECOVERY_LOG utility described in the SCL Advanced File Management Usage manual.

Log entries are not written for the file unless its LOGGING_OPTIONS attribute is ENABLE_MEDIA_RECOVERY. If the LOGGING_OPTIONS attribute is ENABLE_MEDIA_RECOVERY, the default for this parameter is $SYSTEM.AAM.SHARED_RECOVERY_LOG.

NOTE

Whenever you change the LOG_RESIDENCE of an existing keyed-file to a log other than the default log, you must immediately back up the file if you want the entries to be logged. If a back up has not been done since the change and the file is damaged, the RECOVER_FILE_MEDIA subcommand on the RECOVER_KEYED_FILE utility cannot execute successfully for the file.

For more information on logging options, see the SCL Advanced File Management manual and the CYBIL Keyed-File and Sort/Merge Interfaces manual.

RECORD_LIMIT or *RL*

Specifies a new limit for the maximum number of records in each nested file in a keyed file. This new value cannot be less than the old record limit. Omission causes the current maximum number of records to be retained.

RING_ATTRIBUTES or *RING_ATTRIBUTE* or *RA*

Specifies new ring attributes for the file. The parameter values are specified as:

(r1, r2, r3)

The following relation must hold:

minimum ring <= r1 <= r2 <= r3 <= 13

Omission causes the current ring attribute to be retained.

STATEMENT_IDENTIFIER or *SI*

Specifies the length and location of a statement identifier in each record of the file. Omission causes the current statement identifier to be retained.

USER_INFORMATION or *UI*

Specifies a string of 32 characters that is preserved with the file. Omission causes the current user information to be retained.

Remarks

- This command may be used to alter file attributes of permanent and temporary files.
- To alter a file's attributes, the file must not be in use (that is, open) within the job. Furthermore, a permanent file must not be attached to any job other than the requesting job. If attached to the requesting job, the share mode must be NONE.
- You must also have CONTROL permission for the file to alter its file attributes.
- To prevent serious performance degradation, the FORCED_WRITE attribute should be set to FALSE if the LOGGING_OPTIONS attribute includes ENABLE_MEDIA_RECOVERY.
- For more information, see the NOS/VE System Usage manual.

Examples The following example creates a file and changes several attributes.

```
/collect_text $user.fortran_source
ct? program ctime
ct? character*8 time
ct? print*,'The current time is: ',time()
ct? stop
ct? end
ct? **
/change_file_attributes file=$user.fortran_source ..
../file_processor=fortran file_contents=legible
```

CHANGE_IBM_REQUEST Command

Purpose Changes the IBM tape file description in a temporary NOS/VE file formed in a preceding CREATE_IBM_REQUEST command. The IBM tape file must be on an ANSI labeled tape.

Format

CHANGE_IBM_REQUEST or
CHAIR
FILE=**file**
FILE_SET_POSITION=*keyword*
FILE_IDENTIFIER=*string*
FILE_SEQUENCE_NUMBER=*integer*
GENERATION_NUMBER=*integer*
CHARACTER_CONVERSION=*boolean*
STATUS=*status variable*

Parameters

FILE or **F**

Specifies the name of a NOS/VE temporary file associated with an IBM tape file by a previous CREATE_IBM_REQUEST command. This parameter is required.

FILE_SET_POSITION or *FSP*

Specifies the position of the IBM tape file to be read. If you omit this parameter, the NEXT_FILE position is assumed. The parameter can have any of the following values:

BEGINNING_OF_SET or BOS

Specifies that the first tape file on the file set is to be read.

CURRENT_FILE or CF

Specifies that the current tape file is to be read. That is, the last tape file accessed will be accessed again. If the tape is positioned at the beginning of the first volume, the first tape file will be read.

FILE_IDENTIFIER_POSITION or FIP

Specifies that the tape file identified by the FILE_IDENTIFIER and GENERATION_NUMBER parameters is to be read.

FILE_SEQUENCE_POSITION or FSP

Specifies that the tape file identified by the FILE_SEQUENCE_NUMBER parameter is to be read.

NEXT_FILE or NF

Specifies that the tape file following the last accessed tape file will be read. If the tape is positioned at the beginning of the first volume, the first tape file will be read.

FILE_IDENTIFIER or *FI*

Specifies a file identifier as a string of 1 to 17 characters. Each tape file on a multifile set has a unique file identifier. If the FILE_SET_POSITION parameter does not have the FILE_IDENTIFIER_POSITION value, the FILE_IDENTIFIER parameter is ignored.

If you specify the FILE_IDENTIFIER_POSITION value for the FILE_SET_POSITION parameter, the FILE_IDENTIFIER parameter must have a value. If you omit the FILE_IDENTIFIER parameter, a fatal diagnostic is issued.

FILE_SEQUENCE_NUMBER or *FSN*

Specifies the numeric position of a tape file on a multifile set. The position is an unsigned integer in the range 1 through 9999. If the FILE_SET_POSITION parameter is not set to FILE_SEQUENCE_POSITION, the FILE_SEQUENCE_NUMBER parameter value is ignored.

If you specify the FILE_SEQUENCE_POSITION value for the FILE_SET_POSITION parameter, the FILE_SEQUENCE_NUMBER parameter must have a value. If you omit the FILE_SEQUENCE_NUMBER parameter, a fatal diagnostic is issued.

GENERATION_NUMBER or *GN*

Identifies the specific revision of the tape file named by the FILE_IDENTIFIER parameter. The revision is shown as an unsigned integer in the range 1 through 9999. If the FILE_SET_POSITION parameter has the FILE_IDENTIFIER_POSITION value, and the GENERATION_NUMBER parameter is omitted, then the GENERATION_NUMBER parameter value is set to one.

If the FILE_SET_POSITION parameter does not have the FILE_IDENTIFIER_POSITION value, the GENERATION_NUMBER parameter is ignored.

CHARACTER_CONVERSION or *CC*

Boolean value specifies whether or not the tape file data is to be converted to or from its character set. If you omit the CHARACTER_CONVERSION parameter, its value is left at its previous setting.

Of the tape file migration methods, only FMU automatically does character conversion in addition to any conversion specified by the CHARACTER_CONVERSION parameter value.

To obtain a properly migrated tape file, you usually want to set the CHARACTER_CONVERSION parameter to FALSE if you use FMU to migrate. Otherwise you convert your tape file data twice. Set this parameter to TRUE if you use any other tape file migration method. Otherwise, you do not convert your tape file data at all.

Remarks

- This command is handy when you wish to use the same temporary NOS/VE file to reference several tape files from the same multifile set on tape. You simply use the CHANGE_IBM_REQUEST command to change the tape file description.

- In general, when you omit a parameter from the CHANGE_IBM_REQUEST command, the value of that parameter is not changed. Thus, usually, you need to include only those parameters whose settings change for the new IBM tape file.

  The two exceptions to this rule are the FILE_SET_POSITION and the GENERATION_NUMBER parameters. When you omit these parameters, their values become the default values, NEXT_FILE and 1, respectively, for those parameters.

Examples

Suppose you have associated the first tape file of a multifile set on an IBM labeled tape with the temporary NOS/VE file, MULTI_FILES. After you access this first tape file, you want to access the next tape file in the same multifile set. The following command associates the next tape file with MULTI_FILES:

```
/change_ibm_request  file=multi_files ..
../file_set_position=next_file ..
../character_conversion=true
```

The command also says that the data in the next tape file is to be converted from its character set.

CHANGE_INTERACTION_STYLE Command

Purpose Changes the style of system interaction.

Format

CHANGE_INTERACTION_STYLE or
CHAIS
STYLE=keyword
STATUS=status variable

Parameters

STYLE or **S**

Specifies the style of system interaction. You can specify the keyword LINE for line mode or SCREEN for screen mode. This parameter is required.

Remarks

- Interactive parameter prompting is disabled when you first log in. Specifying an interaction style enables interactive parameter prompting.
- The SCL function $INTERACTION_STYLE can be used to determine the current interaction style.
- For more information, see the NOS/VE System Usage manual.

CHANGE_JOB_ATTRIBUTE Command

Purpose Changes the current job's attributes.

Format

CHANGE_JOB_ATTRIBUTE or
CHANGE_JOB_ATTRIBUTES or
CHAJA
COMMENT_BANNER=string
COPIES=integer
CYCLIC_AGING_INTERVAL=integer
DETACHED_JOB_WAIT_TIME=integer or *keyword*
DEVICE=name or *keyword*
DISPATCHING_PRIORITY=name or *keyword*
EXTERNAL_CHARACTERISTICS=string or *keyword*

FORMS_CODE=string or *keyword*
JOB_ABORT_DISPOSITION=keyword
JOB_RECOVERY_DISPOSITION=keyword
MAXIMUM_WORKING_SET=integer or *keyword*
MINIMUM_WORKING_SET=integer
OPERATOR_FAMILY=name
OPERATOR_USER=name
OUTPUT_CLASS=keyword
OUTPUT_DESTINATION=any
OUTPUT_DESTINATION_USAGE=name or *keyword*
OUTPUT_DISPOSITION=file or *keyword*
OUTPUT_PRIORITY=keyword
PAGE_AGING_INTERVAL=integer
REMOTE_HOST_DIRECTIVE=string
ROUTING_BANNER=string
STATION=name or *keyword*
USER_INFORMATION=string
VERTICAL_PRINT_DENSITY=keyword
VFU_LOAD_PROCEDURE=name or *keyword*
STATUS=status variable

Parameters

COMMENT_BANNER or *CB*

Specifies a default character string to be displayed with the output files generated by the job (including the OUTPUT file). Specify a string of 0 to 31 characters. Use of this string is determined by your site.

If omitted, the attribute associated with this parameter does not change.

COPIES or *C*

Specifies the default number of output file copies to be made. The value can be an integer ranging from 1 to 10.

If omitted, this attribute is left unchanged.

CYCLIC_AGING_INTERVAL or *CAI*

Specifies the time, in microseconds, before the memory manager ages the job's working set. This parameter's allowable range is determined by the job's job class.

If omitted, the attribute associated with this parameter does not change.

DETACHED_JOB_WAIT_TIME or *DJWT*

Specifies the number of seconds a job, if detached or disconnected from the terminal session, will remain suspended before the job is terminated.

If you specify UNLIMITED, the job will be suspended indefinitely. by the system will be used. This parameter's allowable range is determined by the job's job class.

If omitted, the attribute associated with this parameter does not change.

DEVICE or *D*

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 you specify AUTOMATIC, the system prints the file at any printer that meets the EXTERNAL_CHARACTERISTICS and FORMS_CODE specifications specified.

If omitted, the attribute associated with this parameter does not change.

DISPATCHING_PRIORITY or *DP*

Specifies the default dispatching priority assigned to all your tasks. Values are P1 (lowest priority) to P10. Attempting to raise the value of this parameter has no effect, even though no error status is returned.

If DEFAULT is specified, the dispatching priority table established by the job's service class is reinstated.

If omitted, the attribute associated with this parameter does not change. A job's initial value for this attribute is determined by its service class.

EXTERNAL_CHARACTERISTICS or *EC*

Specifies a default string to be used by all output files generated by the current job. This string selects a printer having 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.

If this parameter is omitted, this attribute is left unchanged.

FORMS_CODE or *FC*

Specifies a default string for each output file generated by the job. This string selects a printer that has the same string defining 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 you specify NORMAL, the system selects a printer that has a FORMS_CODE value of NORMAL. If you specify NORMAL when the OUTPUT_DESTINATION_USAGE parameter value is DUAL_STATE, the NORMAL value is equivalent to a string of spaces.

If omitted, the attribute associated with this parameter does not change.

JOB_ABORT_DISPOSITION or *JAD*

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.

If omitted, the attribute associated with this parameter does not change.

JOB_RECOVERY_DISPOSITION or *JRD*

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 on the JOB_ ABORT_DISPOSITION parameter is used.

RESTART

Job is automatically resubmitted so that it starts over from the beginning.

TERMINATE

Job is discarded.

MAXIMUM_WORKING_SET or *MAXWS*

Specifies the maximum number of pages in the job's working set. If necessary, the memory manager removes pages from the working set to assure this value is not exceeded. Specify one of the following:

integer

Specifies the maximum number of pages allowed in the job's working set. This value can be any integer between 20 and the maximum value allowed by the job's job class.

SYSTEM_DEFAULT

Specifies that the system default value for this attribute for this job class is to be used. This value may be displayed using the DISPLAY_JOB_ ATTRIBUTE_DEFAULTS command.

UNLIMITED

Specifies that the maximum value allowed by the system is to be used.

If omitted, the attribute associated with this parameter does not change.

MINIMUM_WORKING_SET or *MINWS*

Specifies the minimum number of pages in the job's working set. The memory manager does not remove pages from the working set if doing so reduces the working set to below this value.

This parameter's allowable range is determined by the job's job class.

If omitted, the attribute associated with this parameter does not change.

OPERATOR_FAMILY or *DESTINATION_FAMILY* or *DF* or *OF*

Specifies the default private station or remote system operator family name attribute for output files generated by this job. If the OUTPUT_DESTINATION_USAGE value for an output file is PRIVATE or NTF, 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 user attribute of output files with OUTPUT_DESTINATION_USAGE values of PRIVATE or NTF. If omitted, the attribute associated with this parameter does not change.

OPERATOR_USER or *STATION_OPERATOR* or *SO* or *OU*

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. If omitted, the attribute associated with this parameter does not change.

OUTPUT_CLASS or *OC*

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.

If omitted, the attribute associated with this parameter does not change.

OUTPUT_DESTINATION or *ODE*

Specifies the default location name of the system where the output file is to be sent for printing if the file's OUTPUT_DESTINATION_USAGE attribute is QTF or NTF. For all other values of OUTPUT_DESTINATION_USAGE, this parameter is not meaningful and 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.

If omitted, the attribute associated with this parameter does not change.

OUTPUT_DESTINATION_USAGE or *DESTINATION_USAGE* or *DU* or *ODU*

Specifies the default for either the kind of CDCNET print station where the file is to be printed, or the queue file transfer application to be used to forward the output file to a remote system. The following options are available:

PUBLIC

Indicates that the file is to be printed at a public CDCNET batch I/O station. If this value is specified, the OPERATOR_FAMILY, OPERATOR_USER, OUTPUT_DESTINATION, and REMOTE_HOST_DIRECTIVE attributes are not meaningful.

PRIVATE

Indicates that the file is to be printed at a private CDCNET batch I/O station when the designated station operator is controlling the station. If this value is specified, the OUTPUT_DESTINATION and REMOTE_HOST_DIRECTIVE attributes are not meaningful.

DUAL_STATE

Indicates that the file is to be printed under control of the partner system. If this value is specified, the only meaningful attributes are the FORMS_CODE, COPIES, ROUTING_BANNER, and REMOTE_HOST_DIRECTIVE attributes.

QTF

Indicates that the file is to be forwarded to the remote system identified by the OUTPUT_DESTINATION job attribute for processing by that system.

NTF

Indicates that the file is to be forwarded to a remote NTF system for processing by that system. See your site personnel for more information on NTF.

If this parameter is omitted, this attribute value does not change.

OUTPUT_DISPOSITION or *ODI*

Specifies the default for how the job's standard output is to be disposed. Allowable values are either a file name or one of several keywords. The following list describes the results of each of the allowable values. If omitted, the attribute associated with this parameter does not change.

file_name

Specification of a file name indicates that the standard output is 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 will be sent to the output queue for printing.

DISCARD_ALL_OUTPUT (DAO)

All output generated by the job is to be discarded as it is generated. This includes the standard output 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 to be 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 to 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)

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 for the file name. If the $WAIT_QUEUE subcatalog does not exist at the time the output files are returned, it will be created for the user.

OUTPUT_PRIORITY or *OP*

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

If omitted, the attribute associated with this parameter does not change.

PAGE_AGING_INTERVAL or *PAI*

Specifies the number of CP microseconds that will elapse before the memory manager ages the job's working set.

This parameter's allowable range is determined by the job's job class.

If omitted, the attribute associated with this parameter does not change.

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 parameter to a low value, and the CYCLIC_AGING_INTERVAL parameter to a high

value. Otherwise, set the CYCLIC_AGING_INTERVAL parameter to a low value and the PAGE_AGING_ INTERVAL to a high value.

REMOTE_HOST_DIRECTIVE or *DUAL_STATE_ ROUTE_PARAMETERS* or *DSRP* or *RHD*

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 and output files to remote systems, see the NOS/VE System Usage manual.

If omitted, the current value is not changed.

ROUTING_BANNER or *RB*

Defines a default 0 to 31 character string to be displayed with output files generated by this job. The actual use of this string is determined by the site.

If omitted, the attribute associated with this parameter does not change.

STATION or *S*

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. Note that NTF is not currently supported.

Values can be any valid station name or the keyword AUTOMATIC.

If you specify AUTOMATIC, the system default is used. If omitted, the attribute associated with this parameter does not change.

USER_INFORMATION or *UI*

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.

If omitted, the attribute associated with this parameter does not change.

VERTICAL_PRINT_DENSITY or *VPD*

Specifies the default vertical print density at which the file is to be printed. This value affects 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.

If this parameter is omitted, this attribute does not change.

VFU_LOAD_PROCEDURE or *VLP*

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 parameter 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.

If this parameter is omitted, this attribute value does not change.

Remarks

- The SCL command DISPLAY_JOB_ATTRIBUTE and the SCL function $JOB can be used to determine job attribute values.
- The COMMENT_BANNER, COPIES, DEVICE, EARLIEST_PRINT_TIME, EXTERNAL_CHARACTERISTICS, FORMS_CODE, LATEST_PRINT_TIME, OPERATOR_FAMILY, OPERATOR_USER, OUTPUT_CLASS, OUTPUT_DESTINATION, OUTPUT_DESTINATION_USAGE, OUTPUT_DISPOSITION, OUTPUT_PRIORITY, PURGE_DELAY, REMOTE_HOST_DIRECTIVE, ROUTING_BANNER, STATION, VERTICAL_PRINT_DENSITY, and VFU_LOAD_PROCEDURE parameters on this command establish default values for subsequently executed PRINT_FILE commands.
- Queue file transfers to remote non-NOS/VE systems are not currently supported.
- For more information, see the NOS/VE System Usage manual.

CHANGE_JOB_LIMIT Command

Purpose Changes the job resource limits for a job.

Format

CHANGE_JOB_LIMIT or
CHAJL
NAME=name or keyword
RESOURCE_LIMIT=integer or keyword
STATUS=status variable

Parameters

NAME or N

Name of the resource limit you want to change. You can enter the following keywords:

CP_TIME

Specifies the central processing (CP) time resource limit.

SRU

Specifies the system resource unit (SRU) limit.

TASK

Specifies the system task resource limit.

This parameter is required.

RESOURCE_LIMIT or RL

Specifies a new value for the resource limit. You can enter an integer value or the keyword UNLIMITED. The integer value you enter must not exceed the resource limits allowed for your user name.

This parameter is required.

Remarks

- The SCL command DISPLAY_JOB_LIMIT and the SCL function $JOB_LIMIT can be used to determine a job's resource limits.
- For more information, see the NOS/VE System Usage manual.

CHANGE_KEYED_FILE Command

Purpose Begins a CHANGE_KEYED_FILE utility session.

Format

CHANGE_KEYED_FILE or
CHANGE_KEYED_FILES or
CHAKF
INPUT=file
OUTPUT=file
STATUS=status variable

Parameters

INPUT or **I**

File path of an existing keyed file. If an output file is specified, the input file is opened and copied to the output file and then closed.

This parameter is required.

OUTPUT or *O*

File path of the keyed file to which the input keyed file is copied. The output file must be a duplicate of the input file. If the output file does not exist, the command creates it.

If an output file is specified, only the output file is changed. If OUTPUT is omitted, the input file is changed.

Remarks

- The command utility prompt is:

  ```
  chakf/
  ```

  In response to the chakf/ prompt, you can enter SCL commands and any of these subcommands:

  ADD_RECORDS
  REPLACE_RECORDS
  COMBINE_RECORDS
  EXTRACT_RECORDS
  DELETE_RECORDS
  CREATE_NESTED_FILE
  SELECT_NESTED_FILE
  DELETE_NESTED_FILE
  DISPLAY_NESTED_FILE
  CREATE_ALTERNATE_INDEXES
  HELP
  QUIT

- All subcommands in the session apply to the currently selected nested file. The initially selected nested file is $MAIN_FILE. The nested file selection can be changed by a CREATE_NESTED_FILE or SELECT_NESTED_FILE subcommand.

- If the existing keyed file or a new nested file to be created uses a user-defined collation table, hashing procedure, or compression procedure, the object library containing the compiled table or procedure must be in the program library list before the Change_Keyed_File session begins.

  To add one or more object libraries to the program library list, use the ADD_LIBRARIES parameter on a SET_PROGRAM ATTRIBUTES command. For example:

  ```
  set_program_attributes, add_library=$user.hash_library
  ```

- For more information, see the NOS/VE Advanced File Management Usage manual.

Examples The following session copies an existing keyed file and then ends.

```
/change_keyed_file, input=$user.existing_keyed_file, ..
../output=$user.new_keyed_file
chakf/quit
/
```

CHANGE_LINK_ATTRIBUTES Command

Purpose Changes individual link attributes.

Format

CHANGE_LINK_ATTRIBUTES or
CHANGE_LINK_ATTRIBUTE or
CHALA
FAMILY=any
USER=any
PASSWORD=any
CHARGE=any
PROJECT=any
STATUS=status variable

Parameters

FAMILY or *F*

Specifies the NOS family name. If the partner system is NOS/BE, this parameter has no meaning. Strings specified on this parameter may be from 1 to 31 characters long.

USER or *U*

Specifies the NOS or NOS/BE user name. In NOS/BE, this parameter specifies the name used to access the system and is the default permanent file id if a file id is not specified on subsequent file transfer commands. Strings specified on this parameter may be from 1 to 31 characters long.

PASSWORD or *PW*

Specifies the password used to access either NOS or NOS/BE. Strings specified on this parameter may be from 1 to 31 characters long.

CHARGE or *C*

Specifies the NOS or NOS/BE charge number. Strings specified on this parameter may be from 1 to 31 characters long.

PROJECT or *P*

Specifies the NOS or NOS/BE project number. Strings specified on this parameter may be from 1 to 31 characters long.

Remarks For more information, see the NOS/VE System Usage manual.

CHANGE_LOGIN_PASSWORD Command

Purpose Sets or changes the password that is used to validate your access to the system.

Format

CHANGE_LOGIN_PASSWORD or
SETPW or
SET_PASSWORD or
CHALPW
OLD_PASSWORD=name
NEW_PASSWORD=name
EXPIRATION_INTERVAL=integer or *keyword*
EXPIRATION_DATE=date_time or *keyword*
STATUS=status variable

Parameters

OLD_PASSWORD or *OPW* or *FROM* or *F*

Specifies the name of your current (old) password. This parameter is required if the command is issued within a batch job. NOS/VE prompts for entry of the old and new passwords if the FROM parameter is omitted (both the OLD_PASSWORD and NEW_PASSWORD parameters must be omitted).

NEW_PASSWORD or *NPW* or *TO* or *T*

Specifies the name of your new password. This parameter is required if the command is issued within a batch job. If the command is issued within an interactive job and this parameter is omitted, interactive prompting is given.

EXPIRATION_INTERVAL or *EI*

Specifies the number of days until the password will expire. Note that this value may not exceed the maximum expiration interval set for you by your site.

A value of UNLIMITED indicates that the password will never expire. However, if your maximum expiration interval is anything other than UNLIMITED, use of this value will result in an error.

If both this parameter and the EXPIRATION_DATE parameter are specified on this command, the value specified on the EXPIRATION_DATE parameter is used. However, any use of this parameter causes your expiration interval to be reset to the specified value.

If this parameter is omitted, and the EXPIRATION_DATE parameter is also omitted, the expiration date for the new password is calculated using the expiration interval currently set for your user name.

EXPIRATION_DATE or *ED*

Specifies the date and time the password will expire. Note that the number of days between the current date and the specified expiration date may not exceed the maximum expiration interval set for you by your site.

Enter one of the following values:

YYYY-MM-DD.hh:mm:ss

Date and time the password expires. For instance:

```
1988-01-30.12:53:47
```

Note that the time part of the value is optional. Omission causes 00:00:00 to be used. Thus, to set your expiration date to midnight January 30, 1988, enter:

```
1988-01-30
```

NONE

Indicates that the password will never expire. However, if your maximum expiration interval is anything other than UNLIMITED, use of this value will result in an error.

If this parameter is omitted, the value specified on the EXPIRATION_INTERVAL parameter is used to calculate the expiration date. If that value does not exist, the current expiration interval set for your password is used to calculate the expiration date.

Remarks

- Any valid SCL name constitutes a valid NOS/VE password.
- For NOS dual-state users, your NOS/VE password must match your NOS batch password if you wish to perform dual-state access. This matching requirement does not apply to NOS/BE.
- Your password's expiration date, and your expiration interval and maximum expiration interval can be displayed using the ADMINISTER_VALIDATION command followed by the DISPLAY_USER command. See the LOGIN_PASSWORD field of the resulting output.

- For more information, see the NOS/VE System Usage manual.

Examples In the following example, the CHANGE_LOGIN_PASSWORD command is entered without parameters. The system provides a blacked-out area for secure entry of the old and new passwords.

```
/change_login_password
"system prompts for old and new"
    Enter old password
    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
    Enter new password
    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

In the following example, the old password is provided as a parameter on the CHANGE_LOGIN_PASSWORD command. The system prompts for secure entry of the new password.

```
/change_login_password old_password=pass456
"system prompts for new"
    Enter new password
    XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

In the following example, the CHANGE_LOGIN_PASSWORD command contains both the new and old passwords.

```
/change_login_password old_password=pass456 ..
../new_password=pass789
```

In the following example, an expiration date of July 30, 1988 is specified.

```
/chalpw opw=pass456 npw=pass789 ed=1988-06-30
```

CHANGE_MESSAGE_LEVEL Command

Purpose Selects the form of messages displayed by NOS/VE.

Format

CHANGE_MESSAGE_LEVEL or
SETMM or
SET_MESSAGE_MODE or
CHAML
LEVEL=keyword
STATUS=status variable

Parameters **LEVEL or INFORMATION_LEVEL or IL or L**

Specifies that messages are to be issued in their brief or full form. The keywords are:

BRIEF

Displays messages, including their severity level and message text. Any file references appearing in a message are shown relative to the current working catalog.

FULL

Displays messages, including their severity level, product identifier, condition code, and message text. Any file references appearing in a message are expanded to show their complete path.

This parameter is required.

Remarks For more information, see the NOS/VE System Usage manual.

Examples The following example illustrates the BRIEF and FULL message modes.

```
/change_message_level l=brief
/set_file_attributes data_file
/copy_file data_file output
--ERROR--  FSP$OPEN_FILE was issued for file, DATA_FILE,
which does not exist.
/chaml l=full
/copy_file data_file
--ERROR AM 1016--  FSP$OPEN_FILE was issued for file,
:NVE.$LOCAL.DATA_FILE, which does not exist.
```

A detailed description of a message can be obtained by entering the HELP command.

CHANGE_NATURAL_LANGUAGE Command

Purpose Determines the natural language used for messages and help information. The actual text for most messages and help information must be provided by the user or site.

Format

CHANGE_NATURAL_LANGUAGE or
CHANL
NATURAL_LANGUAGE=name
STATUS=status variable

Parameters

NATURAL_LANGUAGE or *NL*

Specifies the natural language being selected. Valid keywords:

DANISH
DUTCH
ENGLISH
FINNISH
FLEMISH
FRENCH
GERMAN
ITALIAN
NORWEGIAN
PORTUGESE
SPANISH
SWEDISH
US_ENGLISH

You may also specify a name if the natural language you want is not on the list of keywords.

If NATURAL_LANGUAGE is omitted, US_ENGLISH is assumed.

Remarks

- You can use the function $NATURAL_LANGUAGE to return the current natural language.
- For more information, see the NOS/VE System Usage manual.

Examples

The following example sets the natural language of status messages and help information to Spanish.

```
/change_natural_language natural_language=spanish
```

CHANGE_OUTPUT_ATTRIBUTE Command

Purpose Changes the attributes of a file in the output queue.

Format

CHANGE_OUTPUT_ATTRIBUTE or
CHANGE_OUTPUT_ATTRIBUTES or
CHAOA
NAME=list of name
COMMENT_BANNER=string
COPIES=integer
DEVICE=name or *keyword*
EXTERNAL_CHARACTERISTICS=string or *keyword*
FORMS_CODE=string or *keyword*
OPERATOR_FAMILY=name
OPERATOR_USER=name
OUTPUT_CLASS=keyword
OUTPUT_DESTINATION=any
OUTPUT_DESTINATION_USAGE=name or *keyword*
OUTPUT_PRIORITY=keyword
REMOTE_HOST_DIRECTIVE=string
ROUTING_BANNER=string
STATION=name or *keyword*
VERTICAL_PRINT_DENSITY=keyword
VFU_LOAD_PROCEDURE=name or *keyword*
STATUS=status variable

Parameters

NAME or **NAMES** or **N**

Specifies the output file(s) whose attributes you want to change.

Values must be either the system-supplied or user-supplied name. If you specify a user-supplied name, it must be unique. This parameter is required.

COMMENT_BANNER or *CB*

Specifies a character string to be displayed with the printed file. Use of this string is determined by the site.

If omitted, the attribute associated with this parameter does not change.

COPIES or *C*

Specifies the number of copies to be printed. If omitted, the attribute associated with this parameter does not change.

DEVICE or *D*

Specifies a name that, when combined with the STATION parameter value, identifies the printer at which the file is to be printed.

Values 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 specified.

If omitted, the attribute associated with this parameter does not change.

EXTERNAL_CHARACTERISTICS or *EC*

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.

If omitted, the attribute associated with this parameter does not change.

FORMS_CODE or *FC*

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 parameter 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.

If omitted, the attribute associated with this parameter does not change.

OPERATOR_FAMILY or *DESTINATION_FAMILY* or *DF* or *OF*

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 parameter is not meaningful unless the OUTPUT_DESTINATION_USAGE attribute specifies PRIVATE or NTF. If omitted, the attribute associated with this parameter does not change.

OPERATOR_USER or *STATION_OPERATOR* or *SO* or *OU*

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 parameter is not meaningful unless the OUTPUT_DESTINATION_USAGE attribute specifies PRIVATE or NTF.

OUTPUT_CLASS or *OC*

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 one second, and an aging factor of one priority unit per aging interval.

If omitted, the attribute associated with this parameter does not change.

OUTPUT_DESTINATION or *ODE*

Specifies the location name of the system where the output file is to be sent for printing if the file's OUTPUT_DESTINATION_USAGE attribute is QTF or NTF. For all other values of OUTPUT_DESTINATION_USAGE, this parameter is not meaningful and 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.

If omitted, the attribute associated with this parameter does not change.

OUTPUT_DESTINATION_USAGE or *DESTINATION_USAGE* or *DU* or *ODU*

Specifies either the kind of CDCNET print station where the file is to be printed, or the queue file transfer application to be used to forward the output file to a remote system. The following options are available:

PUBLIC

Indicates that the file is to be printed at a public CDCNET batch I/O station. If this value is specified, the OPERATOR_FAMILY, OPERATOR_USER, OUTPUT_DESTINATION, and REMOTE_HOST_DIRECTIVE attributes are not meaningful.

PRIVATE

Indicates that the file is to be printed at a private CDCNET batch I/O station when the designated station operator is controlling the station. If this value is specified, the OUTPUT_DESTINATION and REMOTE_HOST_DIRECTIVE attributes are not meaningful.

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 FORMS_CODE, COPIES, ROUTING_BANNER, and REMOTE_HOST_DIRECTIVE attributes.

QTF

Indicates that the file is to be forwarded to a remote system for printing by that system.

NTF

Indicates that the file is to be forwarded to a remote NTF system for printing by that system. See your site personnel for more information on NTF.

If omitted, the attribute associated with this parameter does not change.

OUTPUT_PRIORITY or *OP*

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

If omitted, the attribute associated with this parameter does not change.

REMOTE_HOST_DIRECTIVE or *DUAL_STATE_ROUTE_PARAMETERS* or *DSRP* or *RHD*

Specifies a default text string which may be used to control output processing of output files. 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.

This parameter is ignored unless the OUTPUT_DESTINATION_USAGE output attribute specify the appropriate value. For more information on submitting output files to remote systems, see the NOS/VE System Usage manual.

If omitted, the attribute associated with this parameter does not change.

ROUTING_BANNER or *RB*

Specifies a character string to be displayed with the printed file. The actual use of this string is determined by the site.

If omitted, the attribute associated with this parameter does not change.

STATION or *S*

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 to be sent.

Values can be any valid station name or the keyword AUTOMATIC. If you specify AUTOMATIC, the system default is used.

If omitted, the attribute associated with this parameter does not change.

VERTICAL_PRINT_DENSITY or *VPD*

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.

If omitted, the attribute associated with this parameter does not change.

VFU_LOAD_PROCEDURE or *VLP*

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.

If omitted, the attribute associated with this parameter does not change.

Remarks

- To use this command, you must be logged in to the NOS/VE system where the files were generated and must be the login user or control user for the files. In addition, your output file must not be printing.
- A file is processed according to the value of its output attributes at the time it leaves the output queue.
- Queue file transfers to non-NOS/VE systems are not currently supported.
- For more information, see the NOS/VE System Usage manual.

CHANGE_SCL_OPTION Command

Purpose Changes options provided by the SCL interpreter.

Format

CHANGE_SCL_OPTION or
CHANGE_SCL_OPTIONS or
CHASCLO or
CHASO
PROMPT_FOR_PARAMETER_CORRECTION=boolean
NAME_FOLDING_LEVEL=keyword
STATUS=status variable

Parameters

PROMPT_FOR_PARAMETER_CORRECTION or *PFPC*

Specifies whether parameter prompting will occur for a command. A value of TRUE causes prompting to occur; FALSE causes prompting to not occur.

Omission causes the current value of this parameter to be left unchanged. The initial value of this parameter is TRUE.

NAME_FOLDING_LEVEL or *NFL*

This parameter specifies which characters allowed in NOS/VE names are considered to be letters with both lower and upper case versions. When a name is specified, any lower case letters in it are folded to their upper case counterparts.

STANDARD_FOLDING

This option designates that only the 26 letters a through z are to be folded (to A through Z).

FULL_FOLDING

This option specifies that in addition to the standard 26 letters, the following characters also are to be folded.

Lower Case Character	Upper Case Character
@	`
[	{
\	|
]	}
^	~

This parameter affects the interpretation of all SCL names, including file names, variable and parameter names, names used within the Source Code Utility and the Object Code Utilities, and so on.

Omission causes this option to be left unchanged. The initial value for this option is STANDARD_FOLDING.

Remarks For more information, see the NOS/VE System Usage manual.

CHANGE_TAPE_LABEL_ATTRIBUTE Command

Purpose Identifies an ANSI file to be read or written on an ANSI labelled tape.

Format

CHANGE_TAPE_LABEL_ATTRIBUTE or
CHANGE_TAPE_LABEL_ATTRIBUTES or
CHATLA
FILE=file
FILE_SET_POSITION=keyword
REWRITE_LABELS=boolean
CREATION_DATE=creation_date
EXPIRATION_DATE=expiration_date

FILE_ACCESSIBILITY_CODE=string
FILE_IDENTIFIER=string
FILE_SEQUENCE_NUMBER=integer
FILE_SET_IDENTIFIER=string
GENERATION_NUMBER=integer
GENERATION_VERSION_NUMBER=integer
BLOCK_TYPE=keyword
CHARACTER_CONVERSION=boolean
CHARACTER_SET=keyword
MAXIMUM_BLOCK_LENGTH=integer
MAXIMUM_RECORD_LENGTH=integer
PADDING_CHARACTER=string
RECORD_TYPE=keyword
STATUS=status variable

Parameters

FILE or **F**

Specifies the name of the NOS/VE magnetic tape file to which the ANSI tape attachment information applies.

The NOS/VE file has a list of magnetic tape volumes associated with the file that was specified with the REQUEST_MAGNETIC_TAPE command entered previously. This parameter is required.

FILE_SET_POSITION or *FSP*

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 entries are:

BEGINNING_OF_SET (BOS)

During a READ operation, this values specifies that the first ANSI file on the file set is to be accessed. During a WRITE operation, this value specifies that the ANSI file to be 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 to be 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 parameters is to be accessed. When writing, this value specifies that the ANSI file identified by these parameters 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 parameter is to be accessed.

END_OF_SET (EOS)

When the REWRITE_LABELS parameter is set to 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 parameter is set to FALSE, this value will cause an error to be returned.

If omitted, NEXT_FILE is assumed.

If the ANSI file identified by the FILE_IDENTIFIER and GENERATION_NUMBER parameters or the FILE_SEQUENCE_NUMBER parameter is not found, the tape is positioned after the last ANSI file on the multifile set.

REWRITE_LABELS or *RL*

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.

If omitted, FALSE is assumed.

CREATION_DATE or *CD*

Specifies the creation date of the ANSI file. Specified in ISO format (yy-mm-dd). If omitted, today's date is assumed.

EXPIRATION_DATE or *ED*

Specifies the lifetime of the ANSI file and, implicitly, the lifetime of any subsequent ANSI files in the volume set. This value 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. If omitted, the system assumes the file has expired.

FILE_ACCESSIBILITY_CODE or *FAC*

Specifies the validation code 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. When reading an ANSI file, the system ignores this parameter.

Values can be any 1-character string consisting of any valid NOS/VE characters. If omitted, ' ' (space) is assumed.

FILE_IDENTIFIER or *FI*

Specifies the label identifier; used to differentiate between ANSI files on a multifile set. If omitted, the leftmost 17 characters of the file path name are assumed.

FILE_SEQUENCE_NUMBER or *FSN*

Specifies the numeric position of an ANSI file on a multifile set. Use this parameter to randomly position the tape to any ANSI file on a multifile set. Values can be any integer from 1 to 9999.

If you specify a value for the FILE_SET_POSITION parameter, this parameter is required; otherwise, the parameter value is ignored.

If omitted during the first access of a file, 1 is assumed. If omitted during subsequent accesses, the last file accessed plus 1 is assumed.

FILE_SET_IDENTIFIER or *FSI*

Specifies a unique identification for a set of ANSI files within an installation. Enter a 1- to 6-character string.

The value specified is used for all subsequent ANSI files written to the file set if this parameter is omitted from subsequent CHANGE_TAPE_LABEL_ATTRIBUTES commands for the same magnetic tape file. If omitted, the VOLUME_IDENTIFIER from the VOL1 label is assumed.

GENERATION_NUMBER or *GN*

Specifies a specific revisison of the ANSI file defined by the FILE_IDENTIFIER parameter. If omitted, 1 is assumed.

GENERATION_VERSION_NUMBER or *GVN*

Specifies the state of processing of the file specified by the FILE_IDENTIFIER and GENERATION_NUMBER parameters. 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. If omitted, 0 is assumed.

BLOCK_TYPE or *BT*

Specifies the NOS/VE block type to be used to access the file. Values can be:

SYSTEM_SPECIFIED (SS)
USER_SPECIFIED (US)

If you are accessing the ANSI file and the REWRITE_LABELS parameter is FALSE, the file's block type is taken from the block type value of the ANSI file's HDR2 label, if the value is present. If the HDR2 value is

absent, the value specified in this parameter is used. If this parameter is omitted and the HDR2 value is absent, the value of the BLOCK_TYPE attribute previously defined for the file is used.

If this parameter is omitted and the REWRITE_LABELS parameter is TRUE, the file attribute BLOCK_TYPE value is used and recorded in the HDR2 label.

CHARACTER_CONVERSION or *CC*

Specifies whether or not file data is to be converted to or from the character set specified by the CHARACTER_SET parameter. If omitted, FALSE is assumed. 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 parameter 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 parameter.

FALSE

No conversion takes place.

If you access a tape file and the REWRITE_LABELS parameter is FALSE, the HDR2 value is used if it is present. If the HDR2 value is absent, the value specified in this parameter is used. If this parameter is omitted and the HDR2 value is absent, the value of the CHARACTER_CONVERSION attribute previously defined for the file is used.

If this parameter is omitted and the REWRITE_LABELS parameter is TRUE, the file attribute CHARACTER_CONVERSION value is used and recorded in the HDR2 label.

CHARACTER_SET or *CS*

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 parameter. If omitted, and the REWRITE_LABELS parameter is TRUE, the file attribute INTERNAL_CODE value is used and recorded in the HDR2 label.

MAXIMUM_BLOCK_LENGTH or *MAXBL*

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 which exceed 4,128 bytes, your site must be configured to allow long tape blocks.

If you are accessing a tape file and the REWRITE_LABELS parameter is FALSE, the maximum block length is taken from the file's HDR2 label, if the value is present. If the HDR2 value is absent, the value specified in this parameter is used. If this parameter is omitted and the HDR2 value is absent, the value of the MAXIMUM_BLOCK_LENGTH attribute previously defined for the file is used.

If this parameter is omitted and the REWRITE_LABELS parameter is TRUE, the file attribute MAXIMUM_BLOCK_LENGTH value is used and recorded in the HDR2 label.

MAXIMUM_RECORD_LENGTH or *MAXRL*

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.

If you are accessing a tape file and the REWRITE_LABELS parameter is FALSE, the maximum record length is taken from the file's HDR2 label, if the value is present. If the HDR2 value is absent, the value specified in this parameter is used. If this parameter is omitted and the HDR2 value is absent, the value of the MAXIMUM_RECORD_LENGTH attribute previously defined for the file is used.

If this parameter is omitted and the REWRITE_LABELS parameter is TRUE, the file attribute MAXIMUM_RECORD_LENGTH value is used and recorded in the HDR2 label.

PADDING_CHARACTER or *PC*

Specifies the NOS/VE padding character used to pad records for ANSI fixed record type (RT=F).

If you are accessing a tape file and the REWRITE_LABELS parameter is FALSE, the padding character is taken from the file's HDR2 label, if the value is present. If the HDR2 value is absent, the value specified in this

parameter is used. If this parameter is omitted and the HDR2 value is absent, the value of the PADDING_CHARACTER attribute previously defined for the file is used.

If this parameter is omitted and the REWRITE_LABELS parameter is TRUE, the file attribute PADDING_CHARACTER value is used and recorded in the HDR2 label.

RECORD_TYPE or *RT*

Specifies the record type used to access the ANSI file. Values are:

FIXED (F)
UNDEFINED (U)
VARIABLE (V)
ANSI_VARIABLE (D)
ANSI_SPANNED (S)

If you are accessing a tape file and the REWRITE_LABELS parameter is FALSE, the record type length is taken from the file's HDR2 label, if the value is present. If the HDR2 value is absent, the value specified in this parameter is used. If this parameter is omitted and the HDR2 value is absent, the value of the RECORD_TYPE attribute previously defined for the file is used.

If this parameter is omitted and the REWRITE_LABELS parameter is TRUE, the file attribute RECORD_TYPE value is used and recorded in the HDR2 label.

Remarks

- Only one ANSI file can be defined for a NOS/VE tape file at a time. A subsequent CHANGE_TAPE_LABEL_ATTRIBUTES command for the same NOS/VE tape file augments previous CHANGE_TAPE_LABEL_ATTRIBUTES commands.
- Before you can use this command you must assign the tape file to your job with a REQUEST_MAGNETIC_TAPE command.
- This command does not cause device assignment to occur. The system rejects this command if the tape file is open at the time you enter the command.

- An ANSI file is considered to be expired on a day whose date is equal to or later than the specified date. To be effective on a multifile set, the expiration date of an ANSI file must be earlier than or equal to the expiration dates of all preceding ANSI files on the set.
- When writing a multifile ANSI labeled tape file, you can use this command to specify a different HDR1 and HDR2 label for each ANSI file written. If used, this command must precede the writing of each ANSI file.
- For more information, see the NOS/VE System Usage manual.

CHANGE_TERMINAL_ATTRIBUTES Command

Purpose Defines and changes the attributes of an interactive terminal.

Format

CHANGE_TERMINAL_ATTRIBUTES or
CHANGE_TERMINAL_ATTRIBUTE or
SETTA or
SET_TERMINAL_ATTRIBUTE or
SET_TERMINAL_ATTRIBUTES or
CHATA
ATTENTION_CHARACTER=string
BACKSPACE_CHARACTER=string
BEGIN_LINE_CHARACTER=string
CANCEL_LINE_CHARACTER=string
CARRIAGE_RETURN_DELAY=integer
CARRIAGE_RETURN_SEQUENCE=string
CHARACTER_FLOW_CONTROL=boolean
CODE_SET=keyword
ECHOPLEX=boolean
END_LINE_CHARACTER=string
END_LINE_POSITIONING=keyword
END_OUTPUT_SEQUENCE=string
END_PAGE_ACTION=keyword
END_PARTIAL_CHARACTER=string
END_PARTIAL_POSITIONING=keyword
FOLD_LINE=boolean
FORM_FEED_DELAY=integer
FORM_FEED_SEQUENCE=string
HOLD_PAGE=boolean

HOLD_PAGE_OVER=boolean
LINE_FEED_DELAY=integer
LINE_FEED_SEQUENCE=string
NETWORK_COMMAND_CHARACTER=string
PAGE_LENGTH=integer
PAGE_WIDTH=integer
PARITY=keyword
PAUSE_BREAK_CHARACTER=string
STATUS_ACTION=keyword
TERMINAL_CLASS=name
TERMINAL_MODEL=name or *keyword*
TERMINATE_BREAK_CHARACTER=string
STATUS=status variable

Parameters

ATTENTION_CHARACTER or *AC*

Specifies the character used to perform the action specified by the ATTENTION_CHARACTER_ACTION connection attribute.

BACKSPACE_CHARACTER or *BC*

Specifies the character used to delete the previous character in an input line.

BEGIN_LINE_CHARACTER or *BLC*

Identifies the character used to indicate the beginning of the line.

CANCEL_LINE_CHARACTER or *CLC*

Specifies the character that, when followed by the END_LINE_CHARACTER, discards the current input line.

CARRIAGE_RETURN_DELAY or *CRD*

Specifies the number of milliseconds the network is to wait before sending additional output to the terminal after a carriage return operation. These characters allow a mechanical printing mechanism to reach the left margin and ensure that characters are not lost due to printing attempts during the carriage return operation.

CARRIAGE_RETURN_SEQUENCE or *CRS*

Defines the sequence of characters that position the cursor or carriage return to the beginning of a line. Values can be a sequence of 0 to 2 characters.

CHARACTER_FLOW_CONTROL or *CFC*

Specifies whether your terminal controls the flow of data using X-ON/X-OFF protocol (DC1 and DC3 characters). Values can be:

TRUE

Uses the X-ON/X-OFF protocol to regulate input and output.

FALSE

Does not use the X-ON/X-OFF protocol.

If this attribute is set the wrong way, you can lose data.

CODE_SET or *CS*

Identifies the code set that your terminal uses (usually ASCII). Values can be:

ASCII

Uses the normal ASCII character set.

TPAPL

Uses the typewriter-pairing ASCII character set with APL print.

BPAPL

Uses the bit-pairing ASCII character set with APL print.

ECHOPLEX or *E*

Specifies whether each character entered on input should be echoed back to the terminal. Values are:

TRUE

Input is echoed to the terminal.

FALSE

Input is not echoed to the terminal.

END_LINE_CHARACTER or *ELC*

Specifies the input character that indicates the end of a complete input line.

HOLD_PAGE_OVER=boolean
LINE_FEED_DELAY=integer
LINE_FEED_SEQUENCE=string
NETWORK_COMMAND_CHARACTER=string
PAGE_LENGTH=integer
PAGE_WIDTH=integer
PARITY=keyword
PAUSE_BREAK_CHARACTER=string
STATUS_ACTION=keyword
TERMINAL_CLASS=name
TERMINAL_MODEL=name or *keyword*
TERMINATE_BREAK_CHARACTER=string
STATUS=status variable

Parameters

ATTENTION_CHARACTER or *AC*

Specifies the character used to perform the action specified by the ATTENTION_CHARACTER_ACTION connection attribute.

BACKSPACE_CHARACTER or *BC*

Specifies the character used to delete the previous character in an input line.

BEGIN_LINE_CHARACTER or *BLC*

Identifies the character used to indicate the beginning of the line.

CANCEL_LINE_CHARACTER or *CLC*

Specifies the character that, when followed by the END_LINE_CHARACTER, discards the current input line.

CARRIAGE_RETURN_DELAY or *CRD*

Specifies the number of milliseconds the network is to wait before sending additional output to the terminal after a carriage return operation. These characters allow a mechanical printing mechanism to reach the left margin and ensure that characters are not lost due to printing attempts during the carriage return operation.

CARRIAGE_RETURN_SEQUENCE or *CRS*

Defines the sequence of characters that position the cursor or carriage return to the beginning of a line. Values can be a sequence of 0 to 2 characters.

CHARACTER_FLOW_CONTROL or *CFC*

Specifies whether your terminal controls the flow of data using X-ON/X-OFF protocol (DC1 and DC3 characters). Values can be:

TRUE

Uses the X-ON/X-OFF protocol to regulate input and output.

FALSE

Does not use the X-ON/X-OFF protocol.

If this attribute is set the wrong way, you can lose data.

CODE_SET or *CS*

Identifies the code set that your terminal uses (usually ASCII). Values can be:

ASCII

Uses the normal ASCII character set.

TPAPL

Uses the typewriter-pairing ASCII character set with APL print.

BPAPL

Uses the bit-pairing ASCII character set with APL print.

ECHOPLEX or *E*

Specifies whether each character entered on input should be echoed back to the terminal. Values are:

TRUE

Input is echoed to the terminal.

FALSE

Input is not echoed to the terminal.

END_LINE_CHARACTER or *ELC*

Specifies the input character that indicates the end of a complete input line.

END_LINE_POSITIONING or *ELP*

Specifies the character string sent to the terminal to position the cursor upon receipt of the END_LINE_CHARACTER. Values are:

CRS

Sends the value of the CARRIAGE_RETURN_SEQUENCE attribute.

LFS

Sends the value of the LINE_FEED_SEQUENCE attribute.

CRSLFS

Sends the value of the CARRIAGE_RETURN_SEQUENCE attribute followed by the value of the LINE_FEED_SEQUENCE attribute.

NONE

Sends no response to the terminal.

END_OUTPUT_SEQUENCE or *EOS*

Defines the sequence of characters that terminates output. Values can be a sequence of 0 to 4 characters.

END_PAGE_ACTION or *EPA*

Specifies whether your terminal divides your output into pages. Values can be either:

FFS

Uses the setting you specified for the FORM_FEED_SEQUENCE attribute.

NONE

Does not take any action.

END_PARTIAL_CHARACTER or *EPC*

Identifies the character which indicates the end of a partial input line. The END_PARTIAL_CHARACTER is not forwarded as part of the data. Values can be any ASCII character.

END_PARTIAL_POSITIONING or *EPP*

Specifies the character string sent to the terminal to position the cursor upon receipt of the END_PARTIAL_CHARACTER. Values can be:

CRS

Uses the setting you specified with the CARRIAGE_RETURN_SEQUENCE attribute.

LFS

Uses the setting you specified with the LINE_FEED_SEQUENCE attribute.

CRSLFS

Uses the setting you sepcified with both the CARRIAGE_RETURN_SEQUENCE and LINE_FEED_SEQUENCE attributes.

NONE

Does not use any character string.

FOLD_LINE or *FL*

Specifies whether the network folds output lines that exceed the PAGE_WIDTH attribute setting. Values are:

TRUE

Folds output lines.

FALSE

Does not fold output lines.

FORM_FEED_DELAY or *FFD*

Increases or decreases the amount of idle time (in milliseconds) between pages of output.

FORM_FEED_SEQUENCE or *FFS*

Defines the sequence of characters that causes a page break (or sets to top of form). This action typically repositions the cursor or paper at the top of another page. Values can be a string of 0 to 7 characters.

HOLD_PAGE or *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. Values are:

TRUE

Terminal output is suspended when a page of output has been displayed.

FALSE

Output is send to the terminal without interruption.

HOLD_PAGE_OVER or *HPO*

Specifies whether the network sends a prompt to the terminal each time a hold page condition occurs. Values are:

TRUE

A prompt is sent to the terminal after a page of output has been displayed.

FALSE

No prompt is sent to the terminal.

LINE_FEED_DELAY or *LFD*

Specifies the number of milliseconds the network is to wait before sending additional output to the terminal after a line feed operation. This allows for a mechanical printing mechanism to reach the left margin and ensures that characters are not lost due to printing attempts during the line feed operation.

LINE_FEED_SEQUENCE or *LFS*

Defines the sequence of characters that indicates a line-feed action. This action moves the cursor down a line or rolls the printer paper up a line in preparation for the next line of information.

Values can be a string of 0 to 2 characters.

NETWORK_COMMAND_CHARACTER or *NCC*

Specifies the character that is used to identify network commands. When this character is the first character in an input line entered at the terminal, the line is processed by the network and is not forwarded to NOS/VE.

PAGE_LENGTH or *PL*

Specifies the number of lines displayed at the terminal as a page of output. If you enter 0 (zero), your output is displayed or printed continuously, regardless of length.

PAGE_WIDTH or *PW*

Specifies the number of characters that the terminal can display on a line. A value of 0 (zero) indicates an infinite page length, meaning that the network does not perform line folding.

PARITY or *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. Values are:

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

Sets parity bit to 1.

NONE

If the INPUT_EDITING_MODE attribute is set to TRANSPARENT, the parity bit passes through unchanged. If set to NORMAL, no parity check is done on input, but the parity bit is set to 0 (zero) in each character sent to the terminal.

ZERO

Sets parity bit to 0 (zero).

PAUSE_BREAK_CHARACTER or *PBC*

Specifies the character used to cause a pause break condition.

STATUS_ACTION or *SA*

Specifies how the network handles status messages from network operators. Values can be any of:

DISCARD (D)

Does not display messages.

HOLD (H)

Saves the last four messages until you change STATUS_ACTION or until your connections end.

SEND (S)

Displays each message when received.

TERMINAL_CLASS or *TC*

Specifies the class of terminal being used. The following classes are defined for NOS/VE:

Keyword	Terminals
TTY	M3x teletypewriters.
C75x	CDC 75x, 722-10, 722-20.
C721	CDC 721.
I2741	IBM 2741.
TTY40	M40 teletypewriters.
H2000	Hazeltine 2000.
X364	ANSI X3.64 terminals, including CDC 722-30.
T4010	Tektronix 4010.
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 terminal.
CDC714_30_40	CDC 714-30 or CDC 714-40.
C711	CDC 711.
CDC714_10_20	CDC 714-10 or CDC714-20.
C73x	CDC 73x.
I2740	IBM 2740.
I3270	IBM 3270.
I3780	IBM 3780.

TERMINAL_MODEL or *TRM* or *TM*

Specifies the name of your terminal model. Currently, this attribute determines what terminal definition is used for full-screen applications such as EDIT_FILE.

TERMINATE_BREAK_CHARACTER or *TBC*

Specifies the character used to cause a terminate break condition.

Remarks

- This command can be used to override the attributes provided NOS/VE by the network.
- Because the terminal attribute values affect all connections from the terminal, use caution if you decide to change any attribute value. In general, terminal attributes should only be changed when you are beginning a terminal session.
- For more information, see the NOS/VE System Usage manual.

Examples The following example changes the cancel line and backspace characters.

```
/change_terminal_attributes ..
../cancel_line_character = '%' ..
../backspace_character = '<'
/"This line will be canceled. % .
*DEL*
"This line uses the new bacspace<<<<<kspace char.
/display_log do=1
10:45:31.003.CI."This line uses the new backspace char.
10:45:59.023.CI.display_log do=1
```

The next example changes the pause break character. When the DISPLAY_LOG command begins executing, the pause break character is entered, followed by the TERMINATE_COMMAND command which is used to terminate the suspended command.

```
/change_terminal_attributes ..
../pause_break_character='?'
/display_log do=3
10:45:31.003.CI."This line uses ? (pause break entered)
*Suspended - 1*
p/terminate_command
Command terminated.
```

The next example defines the terminal model to be a VT 220.

```
/change_terminal_attributes ..
../terminal_model=dec_vt220
```

CHANGE_TERM_CONN_DEFAULTS Command

Purpose Changes the connection attribute defaults for a terminal connection.

Format

CHANGE_TERM_CONN_DEFAULTS or
CHANGE_TERM_CONN_DEFAULT or
CHATCD

ATTENTION_CHARACTER_ACTION=integer
BREAK_KEY_ACTION=integer
END_OF_INFORMATION=string
INPUT_BLOCK_SIZE=integer
INPUT_EDITING_MODE=keyword
INPUT_OUTPUT_MODE=keyword

INPUT_TIMEOUT=boolean
INPUT_TIMEOUT_LENGTH=integer
INPUT_TIMEOUT_PURGE=boolean
PARTIAL_CHARACTER_FORWARDING=boolean
PROMPT_FILE=file
PROMPT_STRING=string
STORE_BACKSPACE_CHARACTER=boolean
STORE_NULS_DELS=boolean
TRANSPARENT_CHARACTER_MODE=keyword
TRANSPARENT_FORWARD_CHARACTER=list of string
TRANSPARENT_LENGTH_MODE=keyword
TRANSPARENT_MESSAGE_LENGTH=integer
TRANSPARENT_TERMINATE_CHARACTER=list of string
TRANSPARENT_TIMEOUT_MODE=keyword
STATUS=status variable

Parameters

ATTENTION_CHARACTER_ACTION or *ACA*

Specifies the type of user interrupt command simulated by the network when the character defined by the ATTENTION_CHARACTER attribute is received from the terminal.

BREAK_KEY_ACTION or *BKA*

Specifies the type of user interrupt command simulated by the network when the break key is pressed at the terminal.

END_OF_INFORMATION or *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.

INPUT_BLOCK_SIZE or *IBS*

Specifies the maximum number of characters (80 to 2000) stored by the network before input data is forwarded to NOS/VE.

INPUT_EDITING_MODE or *IEM*

Specifies how the network edits the data received from the terminal.

INPUT_OUTPUT_MODE or *IOM*

Specifies how the network coordinates the terminal input and output streams.

INPUT_TIMEOUT or *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.

INPUT_TIMEOUT_LENGTH or *ITL*

Specifies the maximum number of milliseconds (0 to 86,401) that a task is to wait for input from a terminal when it reads from a terminal file.

INPUT_TIMEOUT_PURGE or *ITP*

Specifies whether undelivered terminal input and output is to be discarded when an input timeout conditon occurs.

PARTIAL_CHARACTER_FORWARDING or *PCF*

Specifies whether the network forwards a partial input line when an END_PARTIAL_CHARACTER is received from the terminal.

PROMPT_FILE or *PF*

Specifies the local file name of the file to which the prompt string is written.

PROMPT_STRING or *PS*

Specifies the string written to the prompt file when a task reads from the terminal file.

STORE_BACKSPACE_CHARACTER or *SBC*

Specifies how the network processes the character specified as the BACKSPACE_CHARACTER when it is received from the terminal.

STORE_NULS_DELS or *SND*

Specifies whether the network stores or discards the NUL and DEL characters when they are received from the terminal.

TRANSPARENT_CHARACTER_MODE or *TCM*

Specifies the action the network takes when a TRANSPARENT_FORWARD_CHARACTER or a TRANSPARENT_TERMINATE_CHARACTER is received from the terminal.

TRANSPARENT_FORWARD_CHARACTER or *TFC*

Specifies a string of 1 to 4 characters; any of these characters is recognized by the network as the transparent mode forwarding character.

TRANSPARENT_LENGTH_MODE or *TLM*

Specifies the action the network takes when it receives the number of characters specified by the TRANSPARENT_MESSAGE_LENGTH connection attribute.

TRANSPARENT_MESSAGE_LENGTH or *TML*

Specifies the minimum number of characters (1 to 32,767) forwarded in each transparent input message.

TRANSPARENT_TERMINATE_CHARACTER or *TTC*

Specifies a string of 1 to 4 characters; any one of these characters is recognized by the network as the transparent mode terminating character.

TRANSPARENT_TIMEOUT_MODE or *TTM*

Specifies the action the network takes when no input is received from the terminal for an interval of 400 milliseconds or more.

Remarks For more information, see the NOS/VE System Usage manual.

CHANGE_UTILITY_ATTRIBUTES Command

Purpose Changes attributes for a utility initiated by the UTILITY/UTILITYEND command.

Format

CHANGE_UTILITY_ATTRIBUTES or
CHANGE_UTILITY_ATTRIBUTE or
CHAUA
UTILITY=name
ENABLE_SUBCOMMAND_LOGGING=boolean
LINE_PREPROCESSOR=name or *keyword*
ONLINE_MANUAL=name or *keyword*
PROMPT=string
TABLES=file
STATUS=status variable

Parameters

UTILITY or **U**

Specifies the name of the utility whose attributes are to be changed.

ENABLE_SUBCOMMAND_LOGGING or *ESL*

Specifies whether the utility subcommands are logged (YES) or not (NO). The only commands logged are those read from either an interactive file or the main command file ($LOCAL.COMMAND) of a batch job. If this parameter is set to NO, the utility's subcommands are not logged, even if they are read from such a file.

If you omit this parameter, the logging enabled attribute is not changed.

LINE_PREPROCESSOR or *LP*

Reserved.

ONLINE_MANUAL or *OM*

Specifies the online manual that describes the utility. Use the keyword NONE to designate that no online manual is associated with the utility.

If you omit this parameter, the online manual attribute is not changed.

PROMPT or *P*

Specifies the prompt string used for interactive command input. For command lines, a slash character (/) is appended to the specified prompt string. For command continuation lines, the string ../ is appended to the specified prompt string.

If you omit this parameter, the prompt attribute is not changed.

TABLES or *TABLE* or *T*

Specifies the file containing the table of utility subcommands defined by the COMMAND command. This table is searched for subcommands while the utility is active.

If you omit this parameter, the command table for the utility is not changed.

Remarks For more information, see the NOS/VE System Usage manual.

CHANGE_VAX_REQUEST Command

Purpose Changes the VAX tape file description in a temporary NOS/VE file formed in a preceding CREATE_VAX_REQUEST command. The VAX tape file must be on an ANSI labeled tape.

Format

CHANGE_VAX_REQUEST or
CHAVR
FILE=file
FILE_SET_POSITION=keyword
FILE_IDENTIFIER=string
FILE_SEQUENCE_NUMBER=integer
GENERATION_NUMBER=integer
STATUS=status variable

Parameters

FILE or F

Specifies the name of a NOS/VE temporary file associated with a VAX tape file by a previous CREATE_VAX_REQUEST command. This parameter is required.

FILE_SET_POSITION or *FSP*

Specifies the position of the VAX tape file to be read. If you omit this parameter, the NEXT_FILE position is assumed. The parameter can have any of the following values:

BEGINNING_OF_SET or BOS

Specifies that the first tape file on the file set is to be read.

CURRENT_FILE or CF

Specifies that the current tape file is to be read. That is, the last tape file accessed will be accessed again. If the tape is positioned at the beginning of the first volume, the first tape file will be read.

FILE_IDENTIFIER_POSITION or FIP

Specifies that the tape file identified by the FILE_IDENTIFIER and GENERATION_NUMBER parameters is to be read.

FILE_SEQUENCE_POSITION or FSP

Specifies that the tape file identified by the FILE_SEQUENCE_NUMBER parameter is to be read.

NEXT_FILE or NF

Specifies that the tape file following the last accessed tape file will be read. If the tape is positioned at the beginning of the first volume, the first tape file will be read.

FILE_IDENTIFIER or *FI*

Specifies a file identifier as a string of 1 to 17 characters. Each tape file on a multifile set has a unique file identifier. If the FILE_SET_POSITION parameter does not have the FILE_IDENTIFIER_POSITION value, the FILE_IDENTIFIER parameter is ignored.

If you specify the FILE_IDENTIFIER_POSITION value for the FILE_SET_POSITION parameter, the FILE_IDENTIFIER parameter must have a value. If you omit the FILE_IDENTIFIER parameter, a fatal diagnostic is issued.

FILE_SEQUENCE_NUMBER or *FSN*

Specifies the numeric position of a tape file on a multifile set. The position is an unsigned integer in the range 1 through 9999. If the FILE_SET_POSITION parameter is not set to FILE_SEQUENCE_POSITION, the FILE_SEQUENCE_NUMBER parameter value is ignored.

If you specify the FILE_SEQUENCE_POSITION value for the FILE_SET_POSITION parameter, the FILE_SEQUENCE_NUMBER parameter must have a value. If you omit the FILE_SEQUENCE_NUMBER parameter, a fatal diagnostic is issued.

GENERATION_NUMBER or *GN*

Identifies the specific revision of the tape file named by the FILE_IDENTIFIER parameter. The revision is shown as an unsigned integer in the range 1 through 9999. If the FILE_SET_POSITION parameter has the FILE_IDENTIFIER_POSITION value, and the GENERATION_NUMBER parameter is omitted, then the GENERATION_NUMBER parameter value is set to one.

If the FILE_SET_POSITION parameter does not have the FILE_IDENTIFIER_POSITION value, the GENERATION_NUMBER parameter is ignored.

Remarks

- This command is handy when you wish to use the same temporary NOS/VE file to reference several tape files from the same multifile set on tape. You simply use the CHANGE_VAX_REQUEST command to change the tape file description.

- In general, when you omit a parameter from the CHANGE_VAX_REQUEST command, the value of that parameter is not changed. Thus, usually, you need to include only those parameters whose settings change for the new VAX tape file.

The two exceptions to this rule are the FILE_SET_POSITION and the GENERATION_NUMBER parameters. When you omit these parameters, their values become the default values, NEXT_FILE and 1, respectively, for those parameters.

Examples Suppose that you have associated a tape file on a VAX labeled tape with the temporary NOS/VE file, FILE_AGAIN. Once you have accessed this tape file, you wish to access it again. The following command associates the tape file with FILE_AGAIN:

```
/change_vax_request  file=file_again ..
../file_set_position=current_position
```

$CHAR Function

Purpose Returns the ASCII character that corresponds to the integer you specify.

Format

$CHAR
(integer)

Parameters

integer

Specifies the integer from which you want the ASCII character returned. The integer is an ordinal; it represents the position of the character in question within the ASCII collating sequence. This parameter is required.

Remarks For further information about functions, see the NOS/VE System Usage manual.

Examples

- The following example returns the character that is represented as the ASCII code 25 hexadecimal:

```
/display_value $char(25(16))
%
```

- The next example displays the ASCII characters with the decimal ordinals of 65, 66, and 67:

```
/display_value ($char(65),$char(66),$char(67))
A
B
C
```

COBOL Command

Purpose Compiles a COBOL source program.

Format

COBOL
INPUT=file
BINARY_OBJECT=file
LIST=file
LIST_OPTIONS=list of keyword
AUDIT=boolean
BASE_LANGUAGE=keyword
DEBUG_AIDS=list of keyword
DUMP_DATA=boolean
ERROR=file
ERROR_LEVEL=keyword
EXTERNAL_INPUT=file
FED_INFO_PROCESSING_STANDARD=list of keyword
INPUT_SOURCE_MAP=file
LEADING_BLANK_ZERO=boolean
LITERAL_CHARACTER=string
OPTIMIZATION=list of keyword
RUNTIME_CHECKS=list of keyword
STANDARDS_DIAGNOSTICS=list of keyword
SUBPROGRAM=boolean
STATUS=status variable

Parameters

INPUT or *I*

Specifies the file containing the COBOL source statements to be compiled. Default is $INPUT.

BINARY_OBJECT or *BO* or *BINARY* or *B*

Specifies the file to receive the compiled object code. Options are:

Omitted

Same as BINARY=$LOCAL.LGO.

File reference

Writes binary object code to file named by file reference.

$NULL

Does not write any binary object code output.

LIST or *L*

Specifies the file to receive COBOL listable output, including source listing and diagnostics. Options are:

Omitted

Same as L=$LIST.

File reference

Writes listable output to file named by file reference.

$NULL

Does not write listable output to a file.

LIST_OPTIONS or *LO*

Specifies compiler output listing options. You can select multiple options, which are separated by a space or a comma. Options are:

Omitted

Same as LIST_OPTIONS=S.

NONE

Does not select any of the options.

M

Data-name and procedure-name attributes.

O

Object code listing.

R

Cross-reference list of items referenced in the program.

RA

Cross-reference list of all items.

S

Source program listing.

SA

Source program listing, including lines turned off by NOLIST directive.

AUDIT or *AUD* or *A*

Indicates whether the program is being run for the Federal Software Training Center (FSTC) audit testing. Selection of this option also selects the ERROR LEVEL=I and STANDARDS_DIAGNOSTICS=(I,ANSI) parameters. Options are:

Omitted

Does not select AUDIT option.

A=TRUE

Performs FSTC audit testing.

BASE_LANGUAGE or *BL*

The BASE_LANGUAGE parameter allows you to compile a program containing syntax based on different base languages. This is a single value parameter. Valid options are as follows:

Omitted

Equivalent to BL=ANS85.

BL=ANS74

Compiles programs whose syntax is based on the 1974 ANSI COBOL Standard.

BL=ANS85

Compiles programs whose syntax is based on the 1985 ANSI COBOL Standard.

BL=COBOL5

Compiles programs written for compilation by COBOL 5.

DEBUG_AIDS or *DA*

Specifies debugging options. You can select multiple options, which are separated by a space or a coma. Options are:

Omitted

Same as DEBUG_AIDS=NONE.

NONE

Selects no debugging options.

ALL

Selects all debugging options except SY.

DS

Compiles debugging lines in the source program (lines with letter D in column 7).

DT

Generates line number, symbol, and source map loader tables as part of the object code.

OC

Continues producing object code even after it finds source code errors.

SY

Checks the syntax, but does not generate object code. You cannot select this option if you selected the OC option.

TR

Produces flow tracing of all paragraphs executed.

DUMP_DATA or *DD*

Reserved for NOS compatibility.

ERROR or *E*

Specifies the file to receive error information. Default is the file specified by the LIST parameter. If no LIST parameter is specified, default is $ERRORS.

ERROR_LEVEL or *EL*

Indicates the severity of the errors to be printed in the file specified by the ERROR parameter. Options are:

Omitted

Lists all warning, fatal, and catastrophic errors.

NONE

Does not list any errors.

EL=T or EL=I

Lists all trivial, warning, fatal, and catastrophic errors.

EL=W

Lists warning, fatal, and catastrophic errors.

EL=F

Lists fatal and catastrophic errors.

EL=C

Lists catastrophic errors only.

EXTERNAL_INPUT or *EX_INPUT* or *EI*

Specifies the Source Code Utility (SCU) library file to be used for COPY statements. Default is $NULL (no SCU library file).

FED_INFO_PROCESSING_STANDARD or *FIPS*

The FIPS parameter specifies diagnosing of input source statements which do not conform to standards in some part of the 1985 Federal Information Processing Standards (FIPS) COBOL subset. You can specify the entire 1985 FIPS COBOL subset or some part of its optional modules.

The FIPS parameter also permits diagnosing of the syntax identified in the obsolete category of the American National Standard Programming Language COBOL, X3.23-1985. The FIPS parameter has meaning only when BASE_LANGUAGE=ANS85. IF BL=ANS74 OR BL=COBOL5, this parameter is ignored.

This parameter is specified as a list of keyword values. The n that terminates several of the keywords can only be the integer 1 or 2. Specifying two different levels of the same keyword (such as CL1 and CL2) is an error. Valid options are:

Omitted

Equivalent to specifying FIPS=NONE.

CLn

Issues diagnostics for syntax that does not conform to level n of FIPS COBOL for the COMMUNICATIONS optional module. Specifying CLn with OMLn is an error.

DLn

Issues diagnostics for syntax that does not conform to level n of FIPS COBOL for the DEBUG optional module. Specifying DLn with OMLn is an error.

NONE

Does not select any option.

OBSOLETE or O

Issues diagnostics for syntax that is identified in the obsolete category of the 1985 ANSI COBOL standard.

OMLn

Issues diagnostics for syntax that does not conform to level n of FIPS COBOL for all optional modules. Specifying OMLn with CLn, DLn, RWLn, or SLn is an error.

RWLn

Issues diagnostics for syntax that does not conform to level n of FIPS COBOL for the REPORT WRITER optional module. Specifying RWLn with OMLn is an error.

SLn

Issues diagnostics for syntax does not conform to level n of FIPS COBOL for the SEGMENTATION optional module. Specifying SLn with OMLn is an error.

SH

Issues diagnostics for syntax that does conform to the HIGH subset for FIPS COBOL.

SI

Issues diagnostics for syntax that does not conform to the INTERMEDIATE subset for FIPS COBOL.

SM

Issues diagnostics for syntax that does not conform to the MINIMUM subset for FIPS COBOL.

When you specify this parameter, specify the STANDARDS_DIAGNOSTICS parameter to set the severity level of any diagnostics issued.

INPUT_SOURCE_MAP or *ISM*

Specifies the name of the file that contains the source map describing the contents of the source input file. Valid options are:

Omitted

ISM file is constructed during compilation based on the contents of the source input file.

ISM=file reference

ISM is specified by the user. This allows you to specify a file that contains the source map of the input file, such as the OUTPUT_SOURCE_MAP file created by the EXPAND_DECK command of the Source Code Utility (SCU).

LEADING_BLANK_ZERO or *LBZ*

Causes leading blanks in numeric input fields to be treated as zeros during execution. Valid options are:

Omitted

Specifies that numeric fields containing blanks are in error.

LBZ=TRUE

Treats all leading blanks in numeric fields as zeros in arithmetic statements and comparisons.

LITERAL_CHARACTER or *LC*

Changes the character that delimits nonnumeric literals in the source program. Options are "" (apostrophe delimits literals) and ' " ' (quotation mark delimits literals). Default is ' " '.

OPTIMIZATION or *OPTIMIZATION_LEVEL* or *OPT* or *OL*

Specifies the level of optimization to be performed by the compiler. Options are:

Omitted

Same as OPTIMIZATION_LEVEL=LOW.

DEBUG

Produces stylized object code for debugging.

LOW

Produces optimized object code for production runs.

RUNTIME_CHECKS or *RC*

Selects execution-time checking of reference modifiers, subscripts, or index references. Options are:

Omitted

Same as RUNTIME_CHECKS=NONE.

NONE

Performs no runtime checks.

ALL

Selects all options.

R

Checks that reference modifiers fit in the subject data.

S

Checks that subscripts and index references are valid.

STANDARDS_DIAGNOSTICS or *SD*

The STANDARDS_DIAGNOSTICS parameter specifies diagnosing of input source statements that do not conform to American National Standard Programming Language COBOL, X3.23-1974. Valid options are:

Omitted

Same as SD=NONE.

SD=NONE

Does not select any option.

SD=(severity,ANSI)

Specifies that source statements not conforming to the 1985 American National Programming Language COBOL, as specified by the BASE_LANGUAGE and FED_INFO_PROCESSING_STANDARD parameters, are to be diagnosed. When you specify this option, also specify the ERROR_LEVEL parameter and value. Severity can be I, W, or F.

SUBPROGRAM or *SP*

Specifies that the source program is to be compiled as a subprogram, rather than as a main program. Options are TRUE (compiled as a subprogram) and FALSE (compiled as a main program). Default is FALSE.

Remarks For more information, see the COBOL Usage manual.

Examples

```
/COBOL INPUT=$USER.COB_SOURCE ..
../BINARY=COB_OBJ ..
../LIST_OPTIONS=(S,R,M) LIST=COB_LIST
```

This command compiles a COBOL program and selects the following options:

INPUT=$USER.COB_SOURCE

Source statements are read from file $USER.COB_SOURCE.

BINARY=COB_OBJ

Compiled object code is written to file COB_OBJ.

LIST_OPTIONS=(S,R,M)

The compiler produces a source listing, cross-reference map, and attributes map.

LIST=COB_LIST

The listable output selected by the LIST_OPTIONS parameter is written to file COB_LIST.

All other parameters assume default options.

COLLECT_TEXT Command

Purpose Reads lines of text from the command list and writes them to a specified file.

Format

COLLECT_TEXT or
COLT
OUTPUT=file
UNTIL=string
PROMPT=string
SUBSTITUTION_MARK=string or *keyword*
STATUS=status variable

Parameters

OUTPUT or **O**

Identifies the file to which the collected text is to be written and, optionally, specifies how the file is to be positioned prior to use. This parameter is required.

UNTIL or *U*

Specifies the string that terminates text collection. The string does not become part of the file. If omitted, the string '**' is assumed.

PROMPT or *P*

Specifies the prompt string to be issued for each line if input is coming from an interactive terminal. If the null string is specified, no prompt is issued.

The first character in the prompt string is a format effector character. A space character is often used to cause each prompt to be issued on a new line.

Omission of the PROMPT parameter causes ct? to be used.

SUBSTITUTION_MARK or *SM*

Specifies the character used to delimit the text to be substituted, or specifies that no character is used to delimit text.

Corresponding pairs of substitution marks must appear on the same line. If the second substitution mark of a pair is not found on the same line as the first mark, the end of the line is treated as the second substitution mark. If two consecutive marks appear on a line, they are replaced by a single substitution mark in the collected text.

The text between the substitution marks is evaluated as an SCL string expression, the result of which replaces the original text including the substitution marks. If an expression cannot be evaluated or its result cannot be converted to a string, the COLLECT_TEXT command terminates with an error message.

Omission of the SUBSTITUTION_MARK parameter causes NONE to be used.

Remarks

- The text is read from file $COMMAND until a line containing only a termination string is encountered, or an end-of-partition or the end-of-information is encountered.
- The termination string must be the first characters of a line (no leading spaces or other characters).
- For more information, see the NOS/VE System Usage manual.

Examples

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 19, 1987
```

The following listing of a file illustrates how the COLLECT_TEXT command can be used in a batch job to provide input to a compiler. A FORTRAN program is collected into a file named FORTRAN_SOURCE and then read by the FORTRAN compiler.

```
LOGIN USER=SDH PASSWORD=PASS456 FAMILY=NVE
COLLECT_TEXT FORTRAN_SOURCE
      PROGRAM CTIME
      CHARACTER*8 TIME
      PRINT*,'THE CURRENT TIME IS: ',TIME()
      STOP
      END
      **
FORTRAN I=FORTRAN_SOURCE
LGO
LOGOUT
```

The job creates a text file containing a FORTRAN program that displays the current time, calls the FORTRAN compiler to compile the program, executes the program, and logs out.

The final example illustrates the use of substitution marks.

```
/a1='test'
/collect_text output=f1 substitution_mark='@'
ct? line 1
ct? @$substr(a1,1,2)@asing
ct? @@line 3
ct? **
/copy_file f1
line 1
teasing
@line 3
```

COMMAND UTILITY Subcommand

Purpose Declares an entry in a utility command table.

Format

COMMAND
NAME=list of name
PROCESSOR=name
AVAILABILITY=keyword
AUTOMATICALLY_LOG=boolean

Parameters

NAME or NAMES or N

Specifies the name(s) by which the command can be called. This parameter is required.

PROCESSOR or *P*

Specifies the name of the processor for the command.

If you omit this parameter, the first element in the NAME parameter is assumed to be the name of the processor.

AVAILABILITY or *A*

Specifies whether the command appears in a display of the utility's command list entries. Values can be:

ADVERTISED (A)

Command appears in a display of the command list entries.

HIDDEN (H)

Command does not appear in a display of the command list entries.

If you omit this parameter, the command is ADVERTISED.

AUTOMATICALLY_LOG or *AL*

Specifies whether the SCL interpreter should log the command when it is recognized (YES), or leave the logging of the command to the command processor (NO). Designating the command processor as the logging control allows you to specify and restrict access to secure information.

If you omit this parameter, YES is assumed.

Remarks

For more information, see the NOS/VE System Usage manual.

$COMMAND_SOURCE Function

Purpose

Determines the location of the command processor for the requesting command.

Format

$COMMAND_SOURCE

Parameters None.

Remarks

- The returned value is one of the following:
  - File, if the source of the command is a file or catalog.
  - Name of the utility, if the command is a subcommand of a utility.
  - Name $SYSTEM, if the command is a system-supplied command.
- This function is not particularly useful when used in the expression for a parameter to a command because, in this case, it returns the source of the command. Therefore, it is more useful when used in control or assignment statements.
- For further information about functions, see the NOS/VE System Usage manual.

Examples The following example demonstrates the use of $COMMAND_SOURCE in an assignment statement:

```
"The following proc resides on an"
"object library in some catalog."
PROC sample_command
  cs = $command_source
  cat = $path($fname(cs),catalog)
  "The following command executes file"
  "sample_program in the same catalog"
  execute_task $fname(cat//'.sample_program)

PROCEND sample_command
```

COMPARE_FILE Command

Purpose Performs a binary comparison of data on the specified files.

Format

COMPARE_FILE or
COMPARE_FILES or
COMF
FILE=file
WITH=file
ERROR_LIMIT=integer
OUTPUT=file
STATUS=status variable

Parameters

FILE or **F**

Identifies the file to be compared and, optionally, specifies how the file is to be positioned prior to use. This parameter is required.

WITH or **W**

Specifies the file used for comparison and, optionally, specifies how the file is to be positioned prior to use. This parameter is required.

ERROR_LIMIT or *EL*

Specifies the number of comparison errors to display. When this limit is exceeded, the command is terminated. Omission causes 0 (zero) to be used, and no comparison errors are displayed.

OUTPUT or *O*

Specifies the file upon which the comparison errors are displayed and, optionally, specifies how the file is to be positioned prior to use. Omission causes $OUTPUT to be used.

Remarks

- The file attributes are not compared.
- Whenever data on the two files does not match, the position, content, and logical difference is displayed on the specified output file.
- The contents of both files are compared from the open position of each until end-of-information is encountered on the shorter of the two files.
- For more information, see the NOS/VE System Usage manual.

Examples

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 file_1
ct? This is a temporary file
ct? that will be used in a
ct? COMPARE_FILE example.
ct? **
/copy_file file_1 file_2
/edit_file file_2
Begin editing file: $LOCAL.FILE_2
ef/list_forward n=2
ef/replace_text 'x' 'z'
COMPARE_FILE ezample.
ef/end
/compare_file file_1 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 relative words (in hexadecimal) of each file and their logical difference are also shown.

COMPARE_OBJECT_LIBRARY Command

Purpose Compares two object libraries or two object files.

Format

COMPARE_OBJECT_LIBRARY or
COMOL
FILE=file
WITH=file
OUTPUT=file
STATUS=status variable

Parameters

FILE or **F**

Old object file or object library file. This parameter is required.

WITH or **W**

New object file or object library file. This parameter is required.

OUTPUT or *O*

Output file. This file can be positioned. If OUTPUT is omitted, file $OUTPUT is used.

Remarks

- The COMPARE_OBJECT_LIBRARY command does not compare procedure files or text files. If the FILE_CONTENTS attribute of each file is not OBJECT, abnormal status is returned and the files are not compared. Also, the command does not compare an object file with an object library. An attempt to do so returns abnormal status.
- COMPARE_OBJECT_LIBRARY compares the content of the files for the following differences: module content changed, modules deleted, and modules added. If the content of a module has changed, the location where the change begins is displayed.
- COMPARE_OBJECT_LIBRARY does not display differences in module order, module creation date or time, or commentary string within the module header.
- For more information, see the NOS/VE Object Code Management manual.

Examples

The following command compares the library files OLD_LIB and NEW_LIB.

```
/compare_object_library old_lib new_lib
Old modules deleted from OLD_LIB
--------------------------------
SORT1

New modules added to NEW_LIB
----------------------------
SORT2


Modules_changed
---------------

PROC1                        - First difference at record number 0 - LIBRARY
MEMBER HEADER
   REPLACED - 20 20 20 20 20 20 20 20 20  3  2 31 30 3A 32 31 3A 32 39  0
  0  0  0  2 31 31 2F 30 33 2F
       WITH - 20 20 20 20 20 20 20 20 20  3  2 31 30 3A 32 33 3A 31 36  0
  0  2  0  2 31 31 2F 30 33 2F


Number of compare errors:  1
```

$CONDITION_CODE Function

Purpose Returns the code that corresponds to a condition name you specify.

Format

$CONDITION_CODE
(name
keyword)

Parameters

name

Specifies a condition name for which you want the corresponding code. This parameter is required.

keyword

Value that indicates whether you want the condition code returned as an integer or a string. Use one of the following values:

NUMERIC (N)

Specifies an integer from 0 through 0FFFFFFFFF (hexadecimal).

SYMBOLIC (S)

Specifies a string that contains the 2-character product identifier and the condition number from 0 through 0FFFFFF (hexadecimal).

Remarks

- If a condition code for the specified name is not found, a 0 is returned.
- For further information about functions, see the NOS/VE System Usage manual.

Examples

The following example displays the symbolic value for the condition name specified.

```
/display_value $condition_code..
../(cle$alpha_char_in_number,s)
CL 115
```

$CONDITION_NAME Function

Purpose

Returns the condition name that corresponds to the condition code you specify.

Format

$CONDITION_NAME or
$CONDITION
(integer
string)

Parameters

integer

Specifies a code that uniquely identifies the condition whose name you want returned. The default radix is decimal. This parameter is required.

If you specify an integer from 1000000 (hexadecimal) to 0FFFFFFFFFF (hexadecimal), the condition code is completely identified. In this case, you need not use the identifier parameter.

string

Specifies a 2-character product identifier. You must distinguish between uppercase and lowercase letters when specifying the identifier.

This parameter is required only when the condition parameter is an integer from 0 through 1,000,000 (hexadecimal).

Remarks

- The value returned is a string in uppercase letters.
- If no condition name is found for the code you specify, the following string is returned:

  UNKNOWN_CONDITION

- For further information about functions, see the NOS/VE System Usage manual.

Examples

- The following example displays the condition name for an integer condition code expressed in decimal (the default radix):

```
/display_value $condition_name(289037877363)
CLE$ALPHA_CHAR_IN_NUMBER
```

- The next example displays the condition name for a string condition code that includes both an integer and a 2-character product identifier.

```
/display_value $condition_name(115,'CL')
CLE$ALPHA_CHAR_IN_NUMBER
```

CONTINUE
Control Statement

Purpose Exits the current WHEN statement.

Format

CONTINUE
RETRY
WHEN boolean expression

The following forms of the CONTINUE statement are valid:

CONTINUE
CONTINUE *RETRY*
CONTINUE *WHEN boolean condition*
CONTINUE *RETRY WHEN boolean condition*

Parameters *RETRY*

Instructs SCL to return control to the statement that caused the condition. If this parameter is omitted, control is returned to the statement following the statement that caused the WHEN statement.

boolean expression

Specifies whether the exit should be honored. If the expression is TRUE, the exit is taken; if the expression is FALSE, processing continues at the next statement. If this clause is omitted, the next exit is taken.

Remarks

- The following descriptions illustrate the process of exiting with a RETRY 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 statmeent following the one that caused the WHEN statemen t to be executed.

  COMMAND_FAULT or PROGRAM_FAULT, with RETRY

  The statement that caused the WHEN statement to be executed 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.

- For more information, see the NOS/VE System Usage manual.

Examples The following example establishes a condition handler for a pause break (terminal interrupt). The user is prompted for several options whenever an INTERRUPT condition occurs.

```
WHEN interrupt DO
  create_variable name=response kind=string
  LOOP
    accept_line variable=response ..
      prompt='Enter RETRY, CONTINUE, or command:  ' ..
      input=input
    IF $string $translate(tdu,response) = 'RETRY' THEN
      CONTINUE RETRY
    ELSEIF $string($name(response)) = 'CONTINUE' THEN
      CONTINUE
    ELSE
      include_line variable=response
    IFEND
  LOOPEND
WHENEND
```

CONTROL Command

Purpose Use the CONTROL (CON) command to access IM/Control from NOS/VE.

Format

CONTROL or
CON
DICTIONARY=name
USER=name
MODE=keyword
FACILITY=keyword
VIEW=name
INPUT=file
LIST=file
STATUS=status variable

Parameters *DICTIONARY* or *D*

The name of the dictionary to be used, created, or deleted. On the CONTROL command, underscores must be used in place of hyphens. For example, if the dictionary name is ACME-DICTIONARY, you must specify DICTIONARY=ACME_DICTIONARY.

The DICTIONARY parameter is required unless you specify FACILITY=CONTROL on the FACILITY parameter.

USER or *U*

The name by which the caller is known to the dictionary.

On the CONTROL command, underscores must be used in place of hyphens. For example, if the user name is DATA-MGR, you must specify USER=DATA_MGR.

The USER parameter is required unless you specify FACILITY=CONTROL on the FACILITY parameter. If omitted, the user can enter only the control facility and can execute only two commands, CREATE DICTIONARY and QUIT.

MODE or *M*

The mode in which you enter IM/Control commands. The following modes can be specified:

COMMAND (C)

The command mode interface to all IM/Control facilities.

SCREEN (S)

The full screen interface to the dictionary maintenance facility. Screen mode is valid only if you specify FACILITY=DICTIONARY_MAINTENANCE on the FACILITY parameter.

The MODE parameter is optional. For batch use, only command mode is allowed. If omitted during batch use, COMMAND is assumed. If omitted during interactive use, the user is prompted for the mode.

FACILITY or *F*

The IM/Control facility to be used. The following facilities can be specified:

CONTROL (C)
DICTIONARY_MAINTENANCE (DM)
GENERATE (G)
OUTPUT (O)

The FACILITY parameter is required during batch use. If omitted during interactive use, the user is prompted for the facility.

VIEW or *V*

The subset of the data with which this IM/Control session works.

On the CONTROL command, underscores must be used in place of hyphens. For example, if the view name is DBA-VIEW, you must specify VIEW=DBA_VIEW.

The VIEW parameter is optional. If omitted, the default view for the user specified on the USER parameter is used. If you specify FACILITY=CONTROL on the FACILITY parameter, the view is ignored.

INPUT or *I*

The path of a file containing IM/Control commands. These commands are the input to IM/Control.

The INPUT parameter is optional. If omitted during interactive use, $COMMAND is used. If omitted during batch use, IM/Control reads the input from the job stream input.

LIST or *L*

The path of the file to which the IM/Control displayed output is to be written.

The LIST parameter is optional. If omitted, $OUTPUT is used.

Remarks

- For more information, see the IM/Control for NOS/VE Usage manual.

Examples

The following example shows how to access IM/Control in screen mode:

```
/control dictionary=acme_dictionary user=administrator ..
../ mode=screen facility=dictionary_maintenance ..
../ view=admin_view
```

The following example shows the same command in abbreviated form:

```
/control d=acme_dictionary u=administrator m=s f=dm ..
../ v=admin_view
```

CONVERT_APL2_FILE Command

Purpose Converts NOS APL2 workspaces to NOS/VE APL workspace. Used automatically from within NOS/VE APL by starting the workspace :NVE.$SYSTEM.CONVERT_APL2. This can be called from outside NOS/VE APL.

Format

CONVERT_APL2_FILE or
CONAF
FROM=file
TO=file
STATUS=status variable

Parameters *FROM*

Specifies the name of the NOS APL2 file to be converted. If omitted, OLD is assumed.

TO

Specifies the name of the NOS/VE file to which the converted NOS APL2 file is written. If omitted, NEW is assumed.

Remarks

- Files to be converted may reside as permanent files on NOS or as local files on NOS/VE brought over from NOS with the GET_FILE utility (in this case, specify B60 conversion). The resultant file is placed in the specified NOS/VE file catalog.
- When you enter the CONVERT_APL2_FILE command from within the CONVERT_APL2_FILE workspace, you may specify more than one file to be converted. You may enter the command interactively or as a batch job.
- Before APL structured files can be converted from NOS APL2 format to NOS/VE APL format, you must run it through the AFIFIX utility on NOS. This copies the files to get rid of EORs so the files can be transferred to NOS/VE with their file directories in sync with the data. To use AFIFIX, enter:

  ```
  GET AFIFIX/UN=APL0
  ```

  then, enter either of the following:

`AFIFIX,lfn1,lfn2`	Sets only the control byte.
`AFIFIX,NOEOR,lfn1,lfn2-`	Sets the control byte and copies the file without EORs.)

See your local site analyst for more information on AFIFIX.

Examples The following example converts file OLD from NOS APL2 format to NOS/VE APL format and puts the converted file on file NEW in the $USER catalog:

```
CONVERT_APL2_FILE FROM=OLD TO=$USER.NEW
```

This example assumes you have run file OLD through AFIFIX on the NOS side.

CONVERT_APL2_WS Command

Purpose Converts NOS APL2 workspaces to NOS/VE APL workspaces. This is used automatically from within the NOS/VE APL workspace when you start it by entering :NVE.$SYSTEM.CONVERT_APL2. To call it from outside NOS/VE APL, use this command.

Format

CONVERT_APL2_WS or
CONAW
FROM=file
TO=file
CC=boolean
STATUS=status variable

Parameters *FROM*

Specifies the NOS APL file to be converted. If omitted, OLD is assumed.

TO

Specifies the NOS/VE file that will contain the converted file. If omitted, NEW is assumed.

CC or *C170_COMPATIBLE*

Specifies if names of functions in the WFNS workspace are converted to APL/VE special functions or if they are copied over with APL2 definitions. TRUE converts the functions; FALSE does not. If omitted, FALSE is assumed.

Remarks

- Files to be converted may reside as permanent files on NOS or as local files on NOS/VE brought over from NOS with the GET_FILE utility (in this case, specify B60 conversion). The resultant file is placed in the specified NOS/VE file catalog.

- When you enter the CONVERT_APL2_WS command from within the CONVERT_APL2_WS workspace, you may specify more than one file to be converted. You may enter the command interactively or as a batch job.

- Although the workspace is converted to a form which NOS/VE understands, there are differences between NOS APL2 and NOS/VE APL which are not provided for. Two differences, for example, are that the state indicator list is not transferred and NOS file names must be converted to NOS/VE file names. For other differences that must be considered, refer to the APL for NOS/VE Language Definition manual (publication number 60485813).

Examples

The following example converts the user workspace in file OLD of the working catalog from NOS APL2 format to NOS/VE APL format and writes the output to file NEW. Names of functions in the WFNS workspace are converted to NOS/VE APL special functions:

```
CONVERT_APL2_WS FROM=OLD TO=NEW CC=TRUE
```

CONVERT_MODIFY_TO_SCU Command

Purpose Converts a source library file from Modify format to SCU format.

Format

CONVERT_MODIFY_TO_SCU or
CONMTS
OLD_PROGRAM_LIBRARY=file
RESULT=file
LIST=file
NAME_LIST=file
DISPLAY_OPTIONS=keyword
CODE_SET=keyword
KEY=string
STATUS=status variable

Parameters

OLD_PROGRAM_LIBRARY or *OPL*

Modify library file. If OLD_PROGRAM_LIBRARY is omitted, file OPL is used.

RESULT or *R*

SCU library file. If RESULT is omitted, file SOURCE_LIBRARY is used.

LIST or *L*

Listing file. You can specify a file position as part of the file name. If LIST is omitted, file $LIST is used.

NAME_LIST or *NL*

Substitution file. If NAME_LIST is omitted, no names are substituted.

DISPLAY_OPTIONS or *DO*

Indicates the information written on the listing file. Options are:

BRIEF (B)

Brief listing.

FULL (F)

Full listing including the text lines changed by the conversion.

If DISPLAY_OPTIONS is omitted, BRIEF is used.

CODE_SET or *CS*

Indicates the character code set used in the Modify library file. Options are:

ASCII64

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

ASCII612

128-character set (6/12 ASCII using escape codes).

ASCIIMIX

Library contains a mix of decks that use the 64-character and 128-character code sets.

If CODE_SET is omitted, ASCIIMIX is used.

KEY or *K*

One-character string specifying the character used to prefix MODIFY directives and used as the key character on the SCU source library. If KEY is omitted, the character string, '*', is used.

Remarks

- The Modify file can use either 64-character (6-bit display code), or 128-character (6/12 ASCII code), or a mix of 64-character and 128-character set decks.
- The CONVERT_MODIFY_TO_SCU command is a NOS/VE command. Although you can enter the command during an SCU session, it does not affect the working library.
- For more information, see the NOS/VE Source Code Management manual.

Examples

The following command converts the Modify file OPL to an SCU library on file SOURCE_FILE.

A brief report is listed on file $LIST. The names to be substituted are on file NEW_NAMES. OPL uses the 64-character set.

```
/convert_modify_to_scu name_list=new_names
Name conversion list
          * = invalid name. Error if used.

OLD_NAME    NEW_NAME             MODIFICATION_NAME
FTNFORM     FORTRAN_FORMAT       FTNFORM
FTNIO       FORTRAN_IO           FTNIO
FTN=1       FTN_1                FTN_1

Deck list as read from OPL directory

FORTRAN1    FORTRAN2

    2 Decks Converted
  SCU library on file - SOURCE_FILE
```

CONVERT_SCU10_TO_SCU11 Command

Purpose Reads an SCU source library in version 1.0 format and writes it in version 1.1 format.

Format **CONVERT_SCU10_TO_SCU11** or **CONS10TOS11**

BASE=file
RESULT=file
STATUS=status variable

Parameters *BASE* or *B*

Name of the file containing an SCU source library in the version 1.0 library format. If BASE is omitted, an attempt is made to access a file named SOURCE_LIBRARY.

RESULT or *R*

Name of the file to receive the converted library in version 1.1 library format. If RESULT is omitted, the library is written on file SOURCE_LIBRARY.$NEXT.

Remarks For more information, see the NOS/VE Source Code Management manual.

Examples The following command converts the version 1.0 source library file OLD_FORMAT to a version 1.1 source library file named NEW_FORMAT.

```
/convert_scu10_to_scu11 base=old_format ..
../result=new_format
```

CONVERT_SCU170_TO_SCU180 Command

Remarks Reserved for site personnel, Control Data, or future use.

CONVERT_UPDATE_TO_SCU Command

Purpose Converts a source library file from Update format to SCU format.

Format **CONVERT_UPDATE_TO_SCU** or
CONUTS
OLD_PROGRAM_LIBRARY=file
RESULT=file
LIST=file
NAME_LIST=file
DISPLAY_OPTIONS=keyword
CODE_SET=keyword
SELECTION_CRITERIA=file
STATUS=status variable

Parameters *OLD_PROGRAM_LIBRARY* or *OLDPL*

Update library file. If OLD_PROGRAM_LIBRARY is omitted, file OLDPL is used.

RESULT or *R*

SCU library file. If RESULT is omitted, file SOURCE_LIBRARY in your working catalog is used.

LIST or *L*

Listing file. You can specify a file position as part of the file name. If LIST is omitted, file $LIST is used.

NAME_LIST or *NL*

Substitution file. You can specify a file position as part of the file name. If NAME_LIST is omitted, no names are replaced.

DISPLAY_OPTIONS or *DO*

Indicates the information written on the listing file. Options are:

BRIEF (B)

Brief listing.

FULL (F)

Full listing including the text lines changed by the conversion.

If DISPLAY_OPTIONS is omitted, BRIEF is used.

CODE_SET or *CS*

Indicates the character code set used in the Update library file. Options are:

ASCII64

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

ASCII812

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

If CODE_SET is omitted, ASCII812 is used.

SELECTION_CRITERIA or *SC*

Criteria file. You can specify a file position as part of the file name. DEFINE directives from the YANK$$$ deck are converted to selection criteria commands that are written on the file. If SELECTION_CRITERIA is omitted, no selection criteria commands are written.

Remarks

- The Update library file must be in sequential format; it must not be in random format. It must use either 64-character, 6-bit display code or 128-character, 8/12 ASCII code.
- The CONVERT_UPDATE_TO_SCU command is a NOS/VE command. Although you can enter the command during an SCU session, it does not affect the working library.

- For more information, see the NOS/VE Source Code Management manual.

Examples The following command converts the Update library file OLDPL to an SCU library on file SOURCE_LIBRARY. A brief report is listed on file $LIST. The names to be substituted are on file NEW_NAMES. Any DEFINE directives in the file are converted to selection criteria commands written on file OLDPL_CRITERIA.

```
/convert_update_to_scu name_list=new_names ..
../code_set=ascii64 selection_criteria=oldpl_criteria
Name conversion list
          * = invalid name. Error if used.

OLD_NAME    NEW_NAME            MODIFICATION_NAME
FTNFORM     FORTRAN_FORMAT      FTNFORM
FTNIO       FORTRAN_IO          FTNIO
FTN=1       FTN_1               FTN_1

Deck list as read from OLDPL directory

FORTRAN1    FORTRAN2

    2 Decks Converted
  SCU library on file - SOURCE_LIBRARY
```

COPY_FILE Command

Purpose Copies data from one file to another.

Format

COPY_FILE or
COPF
INPUT=file
OUTPUT=file
STATUS=status variable

Parameters *INPUT* or *I*

Identifies the file from which data is to be copied and, optionally, specifies how the file is to be positioned prior to use. Data is copied from the open position until end-of-information (EOI) is reached. Omission causes $INPUT to be used.

OUTPUT or *O*

Specifies the file to which data is to be copied and, optionally, specifies how the file is to be positioned prior to use. Omission causes $OUTPUT to be used.

Remarks

- The copy terminates when COPY_FILE encounters an EOI in the input file. For tape files, the copy terminates when a tape mark is encountered.
- If the input file is empty, COPY_FILE returns an abnormal status condition. If the input file is at its EOI, the exception condition FSE$INPUT_FILE_AT_EOI is returned. If a tape is at a double tape mark, the exception condition AME$INPUT_AFTER_EOI is returned.
- If an unlabeled tape contains sets of data, each followed by a single tape mark, issue COPY_FILE once for each set of data to obtain a complete copy of the tape file.
- This command does not copy single tape marks.
- If the output file has not been registered in a catalog, COPY_FILE creates the output file.
- Unless a SET_FILE_ATTRIBUTE command has been specified for the created output file, this file inherits the file cycle 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 output file's structure may differ from the corresponding attributes of the input file.
- COPY_FILE merges the separate FILE_CONTENTS and FILE_STRUCTURE attribute values into a single FILE_CONTENTS value. It is possible that this merging may truncate the FILE_STRUCTURE value. If this occurs, the warning message FSE$OUTPUT_STRUCTURE_TRUNCATED is issued.
- For more information, see the NOS/VE System Usage manual.

Examples In the following three examples, the first copies the contents of $USER.PROLOG to $LOCAL.A; the second copies file $USER.X to $OUTPUT; and the third copies file $USER.INFILE to file $USER.OUTFILE. In the last example, explicit file positioning is specified, that is, the $BOI file position indicates that file $USER.INFILE is appended to the end of file $USER.OUTFILE.

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

COPY_KEYED_FILE Command

Purpose Performs a record-by-record copy.

Format

COPY_KEYED_FILE or
COPKF
INPUT=list of any
OUTPUT=list of any
PRESERVE_KEY_DEFINITIONS=boolean
STATUS=status variable

Parameters

INPUT or I

File to be copied. You must have at least read permission to the file. This parameter is required.

To specify a nested file, first specify the file reference and then the nested-file name, enclosed in parentheses.

When a nested-file name is not specified, all nested files in the file are copied.

COPY_KEYED_FILE positions the file before the copy according to the open position specified for the file. If a file position is not specified on the file reference, the OPEN_POSITION attribute is used. (The default OPEN_POSITION attribute value is $BOI.)

If the open position is $EOI or $ASIS, only the file attributes are copied.

OUTPUT or *O*

File to which data is copied. You must have at least append permission to the file. This parameter is required.

If the INPUT parameter specifies a nested file, the OUTPUT parameter can specify a nested file. (You cannot copy multiple nested files to a single nested file or to a sequential file.)

To specify a nested file, first specify the file reference and then the nested-file name, enclosed in parentheses.

Do not specify the nested-file name $MAIN_FILE on the OUTPUT parameter when the open position is $BOI. (This requests deletion of $MAIN_FILE which is not allowed.)

PRESERVE_KEY_DEFINITIONS or *PKD*

Indicates whether the alternate-key definitions from the input file (if any) are copied to the output file.

TRUE (ON or YES)

Apply alternate-key definitions.

FALSE (OFF or NO)

Do not apply alternate-key definitions.

If PRESERVE_KEY_DEFINITIONS is omitted, the alternate-key definitions are copied.

Remarks

- The INPUT and OUTPUT parameters cannot specify the same file cycle unless the parameters specify different nested files in the same file.
- COPY_KEYED_FILE supports copying to and from files with sequential and keyed-file organizations. It does not support copying to or from byte-addressable files.
- If the INPUT or OUTPUT file could be shared by more than one instance of open, you should attach the file for exclusive access (SHARE_MODE=NONE) before the copy. This prevents other tasks from locking records which would caused COPY_KEYED_FILE to terminate.

- COPY_KEYED_FILE reads records sequentially using the CYBIL procedure AMP$GET_NEXT. It reads records from the input file until it reads an end-of-partition or end-of-information delimiter.

  As each record is read, COPY_KEYED_FILE writes the record sequentially to the output file using the CYBIL procedure AMP$PUT_NEXT.

- COPY_KEYED_FILE writes statistics to $ERRORS if requested by the respective MESSAGE_CONTROL attributes of the input and output files. It writes the output file statistics before the input file statistics. (For a sequential file, no statistics are written because the MESSAGE_CONTROL attribute has no effect for sequential files.)

- If the output file is a new file (a file that has never been opened), the output file is given the preserved attributes of the input file that have not been defined for the output file.

  Temporary attributes are not copied.

  If no attributes have been defined for the output file (no SET_FILE_ATTRIBUTES commands have been executed for the file), the new output file is given all attributes of the input file with the following exception:

  The RING_ATTRIBUTES attribute of the input file is not given to the output file. The output file is given the RING_ATTRIBUTES attribute of the caller of the COPY_KEYED FILE command.

- When copying to an existing file, the file attributes of the output file are not changed.

- The FILE_CONTENTS attribute value of a keyed file cannot be LIST.

- COPY_KEYED_FILE cannot copy multiple nested files (all nested files in the input file) to a single nested file or to a sequential file.

- COPY_KEYED_FILE creates a new nested file when a nested file name is specified on the OUTPUT parameter and the open position of the output file is $BOI. It creates the nested file with the attributes of the input nested file.
- COPY_KEYED_FILE merges the records of the input nested file with those of the output nested file when the open position of the output file is $ASIS or $EOI.
- COPY_KEYED_FILE requires append, shorten, and modify permissions to delete or create a nested file.
- For more information, see the NOS/VE Advanced File Management Usage manual.

Examples This command copies the keyed file .YOUR.ISFILE to the keyed file $USER.ISFILE. It discards any data or alternate keys on $USER.ISFILE and then copies the data and alternate keys from .YOUR.ISFILE to $USER.ISFILE.

```
/copy_keyed_file .your.isfile $user.isfile
```

This command copies the keyed file $USER.ISFILE to the next cycle of the file. It does not copy the alternate-key definitions.

```
/copkf $user.isfile $user.isfile.$next pkd=no
```

This command copies one nested file to another nested file. If the second nested file does not exist, it is created (identical to the first nested file).

```
/copkf input=($user.direct_access_file, nested_file_1) ..
../output=($user.direct_access_file, nested_file_2)
```

CREATE_170_REQUEST Command

Purpose Creates a NOS/VE temporary file to be associated with a 170 tape file. Future references to the 170 tape file are through the NOS/VE temporary file.

Format

CREATE_170_REQUEST or
CRE1R
FILE=*file*
EXTERNAL_VSN=list of string
RECORDED_VSN=list of string
FILE_SET_POSITION=keyword
FILE_IDENTIFIER=string

FILE_SEQUENCE_NUMBER = integer
GENERATION_NUMBER = integer
INTERNAL_CODE = keyword
CHARACTER_CONVERSION = boolean
BLOCK_TYPE = keyword
RECORD_TYPE = keyword
MAXIMUM_BLOCK_LENGTH = integer
MAXIMUM_RECORD_LENGTH = integer
LABEL_TYPE = keyword
TAPE_FORMAT = keyword
STATUS = status variable

Parameters

FILE or **F**

Specifies the name of a NOS/VE temporary file to be associated with a 170 tape file. This parameter is required.

EXTERNAL_VSN or *EVSN* or *VSN*

Gives the external identification of the tape volume(s) containing the 170 tape file. Each parameter value is a string 1 to 6 characters long.

If you specify more than one external volume serial number (VSN), the volumes are requested in the order specified in the parameter list. If you omit the EXTERNAL_VSN parameter, the RECORDED_VSN parameter is used.

You must specify either the EXTERNAL_VSN parameter or the RECORDED_VSN parameter. Otherwise, a fatal error results.

RECORDED_VSN or *RVSN*

Gives the VSN recorded internally on the ANSI VOL1 label on the tape volume(s) holding the 170 tape file. Each parameter value is a string 1 to 6 characters long. File processing uses the RECORDED_VSN parameter to locate and verify the correct volume.

The RECORDED_VSN parameter is for ANSI labeled tapes only. If you enter this parameter for an unlabeled tape, the parameter is ignored, unless there is no matching EXTERNAL_VSN parameter. In this case the RECORDED_VSN parameter takes on the functions of the EXTERNAL_VSN parameter.

If you specify more than one recorded VSN, the volumes are located and verified in the order specified in the parameter list. If you omit the RECORDED_VSN parameter for an ANSI labeled tape, the system uses the EXTERNAL_VSN parameter to verify the VSN recorded internally on the ANSI VOL1 tape label.

If you specify both the EXTERNAL_VSN and RECORDED_VSN parameters, they are matched; the first external VSN with the first recorded VSN, the second external VSN with the second recorded VSN, and so on. For each such pair, NOS/VE uses the external VSN parameter to direct the system operator to mount the tape with that external VSN. For ANSI labeled tapes, NOS/VE uses the recorded VSN value to verify the VSN recorded internally on the ANSI VOL1 label on that tape.

If there is an EXTERNAL_VSN parameter with no matching RECORDED_VSN parameter, NOS/VE uses the EXTERNAL_VSN parameter to direct the system operator. For ANSI labeled tapes, NOS/VE also uses the EXTERNAL_VSN parameter to verify the VSN recorded internally on the ANSI VOL1 tape label.

If there is a RECORDED_VSN parameter with no matching EXTERNAL_VSN parameter, NOS/VE uses the RECORDED_VSN parameter to direct the system operator. For ANSI labeled tapes, NOS/VE also uses the RECORDED_VSN parameter to verify the VSN recorded internally on the ANSI VOL1 tape label.

FILE_SET_POSITION or *FSP*

Specifies the position of the 170 tape file to be read.

The FILE_SET_POSITION parameter is not needed for unlabeled tapes because NOS/VE assumes that you wish to read the first file on an unlabeled tape. That is, the value of the FILE_SET_POSITION parameter for a tape file on an unlabelled 170 tape is BEGINNING_OF_SET, regardless of what you enter.

Only labeled tapes can use all the values of the FILE_SET_POSITION parameter. If you omit the parameter for a labeled tape, the NEXT_FILE position is assumed.

The parameter can have any of the following values:

BEGINNING_OF_SET or BOS

Specifies that the first tape file on the file set is to be read.

CURRENT_FILE or CF

Specifies that the current tape file is to be read. That is, the last tape file accessed will be accessed again. If the tape is positioned at the beginning of the first volume, the first tape file will be read.

FILE_IDENTIFIER_POSITION or FIP

Specifies that the tape file identified by the FILE_IDENTIFIER and GENERATION_NUMBER parameters is to be read.

FILE_SEQUENCE_POSITION or FSP

Specifies that the tape file identified by the FILE_SEQUENCE_NUMBER parameter is to be read.

NEXT_FILE or NF

Specifies that the tape file following the last accessed tape file will be read. If the tape is positioned at the beginning of the first volume, the first tape file will be read.

FILE_IDENTIFIER or *FI*

Specifies a file identifier as a string of 1 to 17 characters. The FILE_IDENTIFIER parameter is for labeled tapes, and it is ignored if you specify it for an unlabeled tape. Each tape file on a multifile set has a unique file identifier. If you specify the FILE_IDENTIFIER_POSITION value for the FILE_SET_POSITION parameter, the FILE_IDENTIFIER parameter is required; otherwise, its value is ignored.

FILE_SEQUENCE_NUMBER or *FSN*

Specifies the numeric position of a tape file on a multifile set. The position is an unsigned integer in the range 1 through 9999. The FILE_SEQUENCE_NUMBER parameter is for labeled tapes, and it is ignored if you specify it for an unlabeled tape. If you specify the FILE_SEQUENCE_POSITION value for the FILE_SET_POSITION parameter, the FILE_SEQUENCE_NUMBER parameter is required; otherwise, its value is ignored.

GENERATION_NUMBER or *GN*

Identifies a specific revision of the tape file named by the FILE_IDENTIFIER parameter. The revision shows as an unsigned integer in the range 1 through 9999. The GENERATION_NUMBER parameter is for labeled tapes, and it is ignored if you specify it for an unlabeled tape. If the FILE_SET_POSITION parameter has the FILE_IDENTIFIER_POSITION value, and the GENERATION_NUMBER parameter is omitted, then the GENERATION_NUMBER parameter value is set to one.

INTERNAL_CODE or *IC*

Specifies the character set of the data on the tape volume. You can enter one of the following values:

AS6

6/12-bit ASCII

AS8

8/12-bit ASCII

D63

63-character display code

D64

64-character display code

If you omit this parameter, its value is set to D64.

CHARACTER_CONVERSION or *CC*

Specifies with a boolean value whether or not file data is to be converted to or from the character set specified by the INTERNAL_CODE parameter. If you omit the CHARACTER_CONVERSION parameter, FALSE is assumed to be its value.

Of the tape file migration methods, only FMA and FMU automatically do character conversion in addition to any conversion specified by the CHARACTER_CONVERSION parameter value.

To obtain a properly migrated tape file, you usually want to set the CHARACTER_CONVERSION parameter to:

FALSE

if you use FMA or FMU to migrate. Otherwise you convert your tape file data twice.

TRUE

if you use any other tape file migration method. Otherwise, you do not convert your tape file data at all.

BLOCK_TYPE or *BT*

Specifies the block type of the 170 input tape file. Its value can be one of the following keywords:

INTERNAL or I

Internal blocking.

CHARACTER_COUNT or CC

Character count blocking.

If you omit this parameter, its value is set to INTERNAL.

RECORD_TYPE or *RT*

Specifies the record type of the 170 tape file. You can specify one of the follwing record types:

CONTROL_WORD, CW, or W

Control word.

FIXED_LENGTH, FL, or F

A fixed length. record.

SYSTEM_RECORD, SR, or S

System record.

ZERO_BYTE, ZB, or Z

Zero-byte.

If you omit this parameter, the record type is set to CONTROL_WORD.

MAXIMUM_BLOCK_LENGTH or *MAXBL* or *MBL*

Specifies with an unsigned integer the maximum length in 6-bit bytes of a block in the 170 tape file. The system maximum for this parameter is 2,147,483,615. If you omit this parameter, its value is set to 5120.

MAXIMUM_RECORD_LENGTH or *MAXRL* or *MRL*

Specifies with an unsigned integer the maximum length in 6-bit bytes of a record in the 170 tape file. The system maximum for this parameter is 4,398,046,511,103. If you omit this parameter, its value is set to 5120.

LABEL_TYPE or *LT*

Specifies whether the tape is labeled. You can specify one of the following keywords:

LABELLED or L

Same as STANDARD.

STANDARD or S

Tape has standard labels.

UNLABELLED or U

Tape is not labelled.

If this parameter is omitted, its value is set to STANDARD.

TAPE_FORMAT or *TF*

Specifies the tape format of the NOS tape file. If you omit this parameter, its value is set to NOS_INTERNAL. The possible values for this parameter are:

NOS_INTERNAL or NI or I

Internal, NOS default tape format.

NOS_BE_INTERNAL or NBI or SI

SCOPE internal and NOS/BE default tape format.

STRANGER or S

Stranger.

LONG_STRANGER or LS or L

Long block stranger.

Remarks

- See the Migration From NOS to NOS/VE Tutorial/Usage manual or the Migration From NOS/BE to NOS/VE Tutorial/Usage manual for more information about this command.
- You must enter a CREATE_170_REQUEST command before using FMA or FMU to migrate the 170 tape file.

Examples

The following command associates temporary file FILE_FTN with a typical FORTRAN formatted tape file (BLOCK_TYPE=CHARACTER_COUNT, RECORD_TYPE=ZERO_WORD):

```
/create_170_request  file=file_ftn ..
../external_vsn='h20' ..
../recorded_vsn='water' ..
../file_set_position=file_identifier_position ..
../file_identifier='ab_cd_goldfish' ..
../generation_number=3 ..
../internal_code=as6 ..
../character_conversion=true ..
../block_type=character_count ..
../record_type=zero_byte ..
../maximum_block_length=4000 ..
../maximum_record_length=200 ..
../label_type=labelled ..
../tape_format=nos_internal
```

The tape file has the following characteristics:

- Standard labeled volume with external vsn, H20; recorded vsn, WATER.
- Found from its file identifier, AB_CD_GOLDFISH, and generation number, 3.
- Data to be converted from its 6/12-bit ASCII character set.
- Maximum length of a block is 4000 6-bit bytes; of a record, 200 6-bit bytes.
- NOS default tape format.

CREATE_7600_REQUEST Command

Purpose Creates a NOS/VE temporary file to be associated with a 7600 tape file. Future references to the 7600 tape file are through the NOS/VE temporary file.

Format

CREATE_7600_REQUEST or
CRE7R
FILE=file
EXTERNAL_VSN=list of string
RECORDED_VSN=list of string
FILE_SET_POSITION=keyword
FILE_IDENTIFIER=string
FILE_SEQUENCE_NUMBER=integer
GENERATION_NUMBER=integer
INTERNAL_CODE=keyword
CHARACTER_CONVERSION=boolean
BLOCK_TYPE=keyword
RECORD_TYPE=keyword
MAXIMUM_BLOCK_LENGTH=integer
MAXIMUM_RECORD_LENGTH=integer
LABEL_TYPE=keyword
STATUS=status variable

Parameters

FILE or **F**

Specifies the name of a NOS/VE temporary file to be associated with a 7600 tape file. This parameter is required.

EXTERNAL_VSN or *EVSN* or *VSN*

Gives the external identification of the tape volume(s) containing the 170 tape file. Each parameter value is a string 1 to 6 characters long.

If you specify more than one external volume serial number (VSN), the volumes are requested in the order specified in the parameter list. If you omit the EXTERNAL_VSN parameter, the RECORDED_VSN parameter is used.

You must specify either the EXTERNAL_VSN parameter or the RECORDED_VSN parameter. Otherwise, a fatal error results.

RECORDED_VSN or *RVSN*

Gives the VSN recorded internally on the ANSI VOL1 label on the tape volume(s) holding the 7600 tape file. Each parameter value is a string 1 to 6 characters long. File processing uses the RECORDED_VSN parameter to locate and verify the correct volume.

The RECORDED_VSN parameter is for ANSI labeled tapes only. If you enter this parameter for an unlabeled tape, the parameter is ignored, unless there is no matching EXTERNAL_VSN parameter. In this case the RECORDED_VSN parameter takes on the functions of the EXTERNAL_VSN parameter.

If you specify more than one recorded VSN, the volumes are located and verified in the order specified in the parameter list. If you omit the RECORDED_VSN parameter for an ANSI labeled tape, the system uses the EXTERNAL_VSN parameter to verify the VSN recorded internally on the ANSI VOL1 label on the tape.

If you specify both the EXTERNAL_VSN and RECORDED_VSN parameters, they are matched; the first external VSN with the first recorded VSN, the second external VSN with the second recorded VSN, and so on. For each such pair, NOS/VE uses the external VSN parameter to direct the system operator to mount the tape with that external VSN. For an ANSI labeled tape, NOS/VE uses the recorded VSN value to verify the VSN recorded internally on the ANSI VOL1 label on that tape.

If there is an EXTERNAL_VSN parameter with no matching RECORDED_VSN parameter, NOS/VE uses the EXTERNAL_VSN parameter to direct the system operator. For ANSI labeled tapes, NOS/VE also uses the EXTERNAL_VSN parameter to verify the VSN recorded internally on the ANSI VOL1 tape label.

If there is a RECORDED_VSN parameter with no matching EXTERNAL_VSN parameter, NOS/VE uses the RECORDED_VSN parameter to direct the system operator. For ANSI labeled tapes, NOS/VE also uses the RECORDED_VSN parameter to verify the VSN recorded internally on the ANSI VOL1 tape label.

FILE_SET_POSITION or *FSP*

Specifies the position of the 7600 tape file to be read.

The FILE_SET_POSITION parameter is not needed for unlabeled tapes because NOS/VE assumes that you wish to read the first file on an unlabeled tape. That is, the value of the FILE_SET_POSITION parameter for a tape file on an unlabeled 7600 tape is BEGINNING_OF_SET, regardless of what you enter.

Only labeled tapes can use all the values of the FILE_SET_POSITION parameter. If you omit the parameter for a labeled tape, the NEXT_FILE position is assumed.

The parameter can have any of the following values:

BEGINNING_OF_SET or BOS

Specifies that the first tape file on the file set is to be read.

CURRENT_FILE or CF

Specifies that the current tape file is to be read. That is, the last tape file accessed will be accessed again. If the tape is positioned at the beginning of the first volume, the first tape file will be read.

FILE_IDENTIFIER_POSITION or FIP

Specifies that the tape file identified by the FILE_IDENTIFIER and GENERATION_NUMBER parameters is to be read.

FILE_SEQUENCE_POSITION or FSP

Specifies that the tape file identified by the FILE_SEQUENCE_NUMBER parameter is to be read.

NEXT_FILE or NF

Specifies that the tape file following the last accessed tape file will be read. If the tape is positioned at the beginning of the first volume, the first tape file will be read.

FILE_IDENTIFIER or *FI*

Specifies a file identifier as a string of 1 to 17 characters. The FILE_IDENTIFIER parameter is for labeled tapes, and it is ignored if you specify it for an unlabeled tape. Each tape file on a multifile set has a unique file

identifier. If you specify the FILE_IDENTIFIER_POSITION value for the FILE_SET_POSITION parameter, the FILE_IDENTIFIER parameter is required; otherwise, its value is ignored.

FILE_SEQUENCE_NUMBER or *FSN*

Specifies the numeric position of a tape file on a multifile set. The position is an unsigned integer in the range 1 through 9999. The FILE_SEQUENCE_NUMBER parameter is for labeled tapes, and it is ignored if you specify it for an unlabeled tape. If you specify the FILE_SEQUENCE_POSITION value for the FILE_SET_POSITION parameter, the FILE_SEQUENCE_NUMBER parameter is required; otherwise, its value is ignored.

GENERATION_NUMBER or *GN*

Identifies a specific revision of the tape file named by the FILE_IDENTIFIER parameter. The revision shows as an unsigned integer in the range 1 through 9999. The GENERATION_NUMBER parameter is for labeled tapes, and it is ignored if you specify it for an unlabeled tape. If the FILE_SET_POSITION parameter has the FILE_IDENTIFIER_POSITION value, and the GENERATION_NUMBER parameter is omitted, then the GENERATION_NUMBER parameter value is set to one.

INTERNAL_CODE or *IC*

Specifies the character set of the data on the tape volume. If you omit the parameter, its value is set to D64. The INTERNAL_CODE parameter can have the following values:

AS6

6/12-bit ASCII

AS8

8/12-bit ASCII

D63

63-character display code

D64

64-character display code

CHARACTER_CONVERSION or *CC*

Specifies with a boolean value whether or not file data is to be converted to or from the character set specified by the INTERNAL_CODE parameter. If you omit the CHARACTER_CONVERSION parameter, FALSE is assumed to be its value.

Of the tape file migration methods, only FMA and FMU automatically do character conversion in addition to any conversion specified by the CHARACTER_CONVERSION parameter value.

To obtain a properly migrated tape file, you usually want to set the CHARACTER_CONVERSION parameter to FALSE if you use FMA or FMU to migrate. Otherwise you convert your tape file data twice. Set it to TRUE if you use any other tape file migration method. Otherwise, you do not convert your tape file data at all.

BLOCK_TYPE or *BT*

Specifies the block type of the 7600 input tape file. If you omit this parameter, its value is set to INTERNAL. Its value can be either of the following:

INTERNAL or I

Internal blocking

CHARACTER_COUNT or CC

Character count blocking

RECORD_TYPE or *RT*

Specifies the record type of the 7600 tape file. If you omit this parameter, its value is set to CONTROL_WORD. Its value can be any of the following:

CONTROL_WORD, CW, or W

Control word

FIXED_LENGTH, FL, or F

Fixed length

SYSTEM_RECORD, SR, or S

System record

ZERO_BYTE, ZB, or Z

Zero-byte

MAXIMUM_BLOCK_LENGTH or *MAXBL* or *MBL*

Specifies with an unsigned integer the maximum length in 6-bit bytes of a block in the 7600 tape file. The system maximum for this parameter is 2,147,483,615. If you omit this parameter, its value is set to 5120.

MAXIMUM_RECORD_LENGTH or *MAXRL* or *MRL*

Specifies with an unsigned integer the maximum length in 6-bit bytes of a record in the 7600 tape file. The system maximum for this parameter is 4,398,046,511,103. If you omit this parameter, its value is set to 5120.

LABEL_TYPE or *LT*

Specifies whether the tape is labeled. If you omit this parameter, its value is set to STANDARD. This parameter can have the following values:

LABELLED or L

Same as STANDARD.

STANDARD or S

Tape has standard labels.

UNLABELLED or U

Tape is not labeled.

Remarks

- This command is very similar to the CREATE_170_REQUEST_command. For more information about the CREATE_170_REQUEST command, see the Migration From NOS to NOS/VE Tutorial/Usage manual or the Migration From NOS/BE to NOS/VE Tutorial/Usage manual.
- You must enter a CREATE_7600_REQUEST command before using FMA or FMU to migrate the 7600 tape file.

Examples The following command associates temporary file FILE_FTN with a typical FORTRAN unformatted tape file (BLOCK_TYPE=INTERNAL, RECORD_TYPE=CONTROL_WORD):

```
/create_7600_request  file=file_ftn ..
../external_vsn='agua' ..
../recorded_vsn='wasser' ..
../file_set_position=file_sequence_position ..
../file_sequence_number=7 ..
../internal_code=d64 ..
../character_conversion=false ..
../block_type=internal ..
../record_type=control_word ..
../maximum_block_length=5120 ..
../maximum_record_length=5120 ..
../label_type=standard
```

The tape file has the following characteristics:

- Volume with external vsn, AGUA; recorded vsn, WASSER.
- Found from its position, seventh, within the multifile set.
- Other parameters have their default values.

CREATE_ALTERNATE_INDEXES Command

Purpose Initiates execution of the CREATE_ALTERNATE_INDEXES command utility. The utility can create, delete, and display alternate-key definitions in a keyed file.

Format

CREATE_ALTERNATE_INDEXES or
CREATE_ALTERNATE_INDEX or
CREATE_ALTERNATE_INDICES or
CREAI
INPUT=list of any
STATUS=status variable

Parameters

INPUT or I

Keyed file to be processed by the utility. The file permissions required depend on the subcommands entered during the utility as described in the Remarks. This parameter is required.

To specify a nested file, first specify the file reference and then the nested-file name, enclosed in parentheses.

Remarks

- The command utility prompt is:

```
creai/
```

- In response to the creai/ prompt, you can enter NOS/VE commands and any of these subcommands:

  QUIT
  DISPLAY_KEY_DEFINITIONS
  CREATE_KEY_DEFINITION
  DELETE_KEY_DEFINITION
  CANCEL_KEY_DEFINITIONS
  APPLY_KEY_DEFINITIONS

- The CREATE_ALTERNATE_INDEXES utility creates the specified keyed file if:

  - The file does not exist and,
  - A SET_FILE_ATTRIBUTES command has specified the KEY_LENGTH and MAXIMUM_RECORD_LENGTH attributes for the file.

  If the SET_FILE_ATTRIBUTES command defining the new file omits an attribute, the default attribute value is used. However, if it omits the FILE_ORGANIZATION attribute, indexed-sequential organization is used.

- The CREATE_ALTERNATE_INDEXES command does not check your file permissions. The subcommands you enter in the utility session check that you have the required permissions to do the operation.

  To display key definitions, you must have at least read permission. To create, delete, cancel, or apply key definitions, you must have at least three permissions: append, modify, and shorten.

- For more information, see the NOS/VE Advanced File Management Usage manual.

Examples This command begins a utility session that displays the alternate key definitions of keyed file $USER.IS_FILE.

```
/create_alternate_indexes input=$user.is_file
crea1/display_key_definitions key_names=all display_options=brief
        Display_Key_Definitions        NOS/VE Keyed File Utilities 1.1
File = :NVE.USER99.IS_FILE

 KEY NAME                  POSITION    LENGTH   TYPE         STATE
 ------------------------ --------    ------   ----------   --------------
 ALTERNATE_KEY_1                0        10   uncollated   Exists in file
crea1/quit      "The APPLY_KEY_DEFINITIONS parameter is not required here"
                "because no creation or deletion requests are pending."
```

CREATE_CATALOG Command

Purpose Creates a new catalog.

Format

CREATE_CATALOG or
CREC
CATALOG=file
STATUS=status variable

Parameters

CATALOG or C

Specifies the catalog to be created. This parameter is required.

Remarks

- An empty catalog is defined and is registered in the catalog you specify.
- Only the master catalog owner can create a new catalog.
- For more information, see the NOS/VE System Usage manual.

Examples The following example creates the catalog CATALOG_2 in the master catalog.

```
/create_catalog $user.catalog_2
/disc $user
 CATALOG: CATALOG_1
 CATALOG: CATALOG_2
    FILE: DATA_FILE_1
    FILE: EPILOG
    FILE: PROLOG
```

CREATE_CATALOG_PERMIT Command

Purpose Establishes or modifies an access control entry for a catalog.

Format

CREATE_CATALOG_PERMIT or
CRECP
CATALOG=file
GROUP=keyword
FAMILY_NAME=name
USER=name
ACCOUNT=name
PROJECT=name
ACCESS_MODES=list of keyword
SHARE_MODES=list of keyword
APPLICATION_INFORMATION=string
STATUS=status variable

Parameters

CATALOG or **C**

Specifies the catalog for which an access control entry is being established or modified. This parameter is required.

GROUP or *G*

Specifies if the permit entry is for a specific user or a group of users. The selections are:

PUBLIC

The permit entry applies to all users regardless of family, account, project, or user identifications.

FAMILY

The permit entry applies to all users in the specified family.

ACCOUNT

The permit entry applies to all users associated with the specified family and account identifications.

PROJECT

The permit entry applies to all users associated with the specified family, account, and project identifications.

USER

The permit entry applies to the user identified by the specified family and user identifications.

USER_ACCOUNT

The permit entry applies to the user identified by the specified family, account, and user identifications.

MEMBER

The permit entry applies to the user identified by the specified family, account, project, and user identifications.

Omission causes USER to be used.

FAMILY_NAME or *FN*

Specifies the family name to be permitted access. Omission causes the family name associated with the requesting job to be used if the GROUP selection indicates this parameter is applicable. If the GROUP selection indicates this parameter is not applicable and this parameter is specified, an abnormal status is returned.

USER or *U*

Specifies the user name to be permitted access. Omission causes the user name associated with the requesting job to be used if the GROUP selection indicates this parameter is applicable. If the GROUP selection indicates this parameter is not applicable and this parameter is specified, an abnormal status is returned.

ACCOUNT or *A*

Specifies the account to be permitted access. Omission causes the account associated with the requesting job to be used if the GROUP selection indicates this parameter

is applicable. If the GROUP selection indicates this parameter is not applicable and this parameter is specified, an abnormal status is returned.

PROJECT or *P*

Specifies the project to be permitted access. Omission causes the project associated with the requesting job to be used if the GROUP selection indicates this parameter is applicable. If the GROUP selection indicates this parameter is not applicable and this parameter is specified, an abnormal status is returned.

ACCESS_MODES or *ACCESS_MODE* or *AM*

Specifies how all files registered relative to the catalog may be used by the group specified. The access modes are:

READ

Access is permitted to read files.

APPEND

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

MODIFY

Access is permitted to alter data within existing files.

EXECUTE

Access is permitted to execute object code or SCL procedures in the files.

SHORTEN

Access is permitted to delete data from the end of files.

WRITE

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

ALL

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

CONTROL

Access is permitted to delete files and to change file identifications and/or attributes.

CYCLE

Access is permitted to add new file cycles and to create the initial cycle of files.

NONE

Access is specifically prohibited.

Omission causes READ and EXECUTE to be used.

SHARE_MODES or *SHARE_MODE* or *SM*

Specifies how all files registered relative to the catalog must be shared, as a minimum, by the group specified. The share modes are:

READ

Permitted jobs are required to share the file for READ access.

APPEND

Permitted jobs are required to share the file for APPEND access.

MODIFY

Permitted jobs are required to share the file for MODIFY access.

SHORTEN

Permitted jobs are required to share the file for SHORTEN access.

WRITE

Permitted jobs are required to share the file for SHORTEN, MODIFY, and APPEND access.

EXECUTE

Permitted jobs are required to share the file for EXECUTE access.

ALL

Permitted jobs are required to share the file for READ, APPEND, MODIFY, SHORTEN, WRITE, and EXECUTE access.

NONE

No sharing requirements are imposed on permitted jobs. When attaching the files registered in this catalog, permitted jobs may select exclusive access or any of the READ, APPEND, MODIFY, SHORTEN, WRITE, or EXECUTE share modes.

Omission causes the share mode to be determined by the value specified on the ACCESS_MODE parameter of this command. If access mode includes APPEND, SHORTEN, MODIFY, or WRITE, the share requirement chosen is NONE. Otherwise, the share requirements chosen are READ and EXECUTE.

APPLICATION_INFORMATION or *AI*

Specifies information to be saved in the permit entry for use by application programs for additional access controls they impose. A string of up to 31 characters can be specified. Omission causes a string of spaces to be used.

Remarks

- This request allows the catalog owner to specify a general permission that applies to all attempts to access any files registered relative to the specified catalog.
- Permissions can be established for specific users or for groups of users associated with a specific family, account, and project.
- If an access control entry already exists for the specified user or group of users, then the specified access modes, share modes, and application information replace the values currently defined.
- To display a catalog permit, use the DISPLAY_CATALOG command with the DISPLAY_OPTIONS parameter.
- This permit applies to all files in the catalog, except those for which a file permit exists for a group that includes the user who is attempting to gain access.

- For more information, see the NOS/VE System Usage manual.

Examples In the following example, user KRJ is given permission to catalog CATALOG_1 in the master catalog.

```
/create_catalog_permit $user.catalog_1 group=user ..
../user=krj am=write sm=none
/disc $user.catalog_1 permits
   PERMIT_GROUP: USER
     FAMILY: NVE, USER: KRJ
     PERMITS: SHORTEN, APPEND, MODIFY
     SHARE: NONE
     APPLICATION_INFORMATION:
```

In this example, catalog permit allows KRJ append, modify, and shorten access to all files in catalog CATALOG_1. User KRJ is not restricted to any share mode.

This permit applies to all files in the catalog, except those for which a file permit exists for a group that includes the user who is attempting to gain access.

CREATE_COMMAND_LIST_ENTRY Command

Purpose Adds entries to either the beginning or the end of the command list.

Format

CREATE_COMMAND_LIST_ENTRY or
CREATE_COMMAND_LIST_ENTRIES or
CRECLE
ENTRY=list of file or keyword
PLACEMENT=keyword
STATUS=status variable

Parameters

ENTRY or ENTRIES or E

Specifies the entries to be added to the command list. If this parameter is specified as $SYSTEM, the $SYSTEM command library is added to the command list. This parameter is required.

PLACEMENT or *P*

Specifies whether the entries added to the command list are placed before or after the current entries. Use one of the following keywords:

AFTER (A)

Causes the new entries to be placed after the current entries.

BEFORE (B)

Causes the new entries to be placed before the current entries.

Omission of this parameter causes BEFORE to be used.

Remarks

- This command may not be used when the command list search mode is EXCLUSIVE.
- If the command list search mode is RESTRICTED, AFTER is the only allowable value for the PLACEMENT parameter.
- For more information, see the NOS/VE System Usage manual.

CREATE_FILE Command

Purpose Creates a new file, or file cycle and attaches the cycle to the job.

Format

CREATE_FILE or
CREF
FILE=file
LOCAL_FILE_NAME=name
PASSWORD=name or *keyword*
RETENTION=integer
LOG=boolean
STATUS=status variable

Parameters **FILE or F**

Specifies the path and cycle reference of the file to be created. It is also used to direct the registration in a catalog. The file name and cycle must be unique within the catalog in which the file is to be registered, or the command is terminated with an error status.

Omission of a cycle causes $NEXT to be used (the initial cycle is numbered 1). This parameter is required.

LOCAL_FILE_NAME or *LFN*

Specifies a local file name (an alias) which can be used by subsequent commands and programs within the job to refer to the file. If this name is already assigned within the job, an error status is returned. Omission causes the permanent file name to be used.

NOTE

Each attach within a job requires a unique LOCAL_FILE_NAME. For this reason and for compatability with future NOS/VE releases, it is recommended that you specify a unique LOCAL_FILE_NAME value. Furthermore, it is recommended that you not create an SCL variable with the same name as the LOCAL_FILE_NAME. For example:

```
/create_variable n=lfn k=string value=$unique
/create_file f=$user.file local_file_name=$name(lfn1)
```

PASSWORD or *PW*

Specifies the password. A CREATE_FILE of an initial file cycle results in saving this parameter with the catalog entry. This password must then be specified with subsequent requests to access any cycle of the file. For a CREATE_FILE of an additional cycle, this value must match the existing file password. Omission or specification of the keyword value NONE causes no password to be used.

RETENTION or *R*

Specifies the number of days (from 1 to 999) from the current date that the file cycle is to be retained. The number 999 indicates an infinite retention period. Omission causes 999 to be used.

LOG or *L*

Specifies whether the system should keep a log of file access activity. If TRUE is selected, the system maintains a unique log entry for each user who accesses any cycle of the file. This parameter is used only when defining an initial file cycle. Omission causes FALSE to be used.

Remarks

- This command records the file name, password, and log selections in a catalog entry when defining the initial file cycle.

  A unique cycle descriptor that contains the cycle number of the new file cycle, its creation date and time, last access date and time, last modification date and time, expiration date, and a set of usage statistics is also recorded in the catalog.

- In addition to the catalog registration, this command attaches the file for access within the requesting job.

- Ordinarily, it is not necessary to use this command to reference a file. NOS/VE will automatically attach a file when it is referenced by a command. However, you may need to use this command in the following cases:

  - The file has a password
  - The subsequent commands require the use of a local file name due to programming language conventions.
  - The file may have multiple cycles and you did not provide a cycle number in the file reference. In this case you may need to use a local file name to ensure that subsequent commands consistently reference the file cycle you attached.
  - The subsequent commands each read part of the file using the $ASIS open position.

- For more information, see the NOS/VE System Usage manual.

Examples The following example creates file DATA_FILE_1 in the master catalog, a password of PW_FOR_DATA_FILE_1, a log selection of TRUE, and a retention period of 30 days.

```
/create_file $user.data_file_1 pw=pw_for_data_file_1 ..
../r=30 log=true
/detach_file $user.data_file_1
```

The following example creates another cycle of file DATA_FILE_1 in the master catalog. A cycle number of 88 is specified.

```
/cref $user.data_file_1.88 pw=pw_for_data_file_1
/detach_file $user.data_file_1.88
```

The following example creates a file named DATA_FILE_1 in subcatalog CATALOG_1 (refer to the CREATE_CATALOG command) of the user's master catalog. An initial cycle of 1 is assumed.

```
/create_catalog $user.catalog_1
/create_file $user.catalog_1.data_file_1
/detach_file $user.catalog_1.data_file_1
```

Since the first CREATE_FILE command created cycle 1, the next cycle created must be a cycle other than 1. The following command creates cycle 2 of the same file.

```
/create_file $user.catalog_1.data_file_1.2
/detach_file $user.catalog_1.data_file_1.2
```

The following example creates a second file in subcatalog CATALOG_1.

```
/create_file $user.catalog_1.data_file_2 password=..
../data_file_2_password retention=3 log=false
/detach_file $user.catalog.1_data_file_2
```

CREATE_FILE_CONNECTION Command

Purpose Creates a connection between one file (the subject) and another file (the target) so that any data access request against the subject file is passed on to the target file.

Format

CREATE_FILE_CONNECTION or
CREFC
STANDARD_FILE=file
FILE=file
STATUS=status variable

Parameters

STANDARD_FILE or SF

Specifies either one of the following file names: $ECHO, $ERRORS, $INPUT, $LIST, $OUTPUT, $RESPONSE or any other file. This parameter is required.

FILE or F

Specifies the name of the target file to be connected to the specified subject file. This parameter is required.

Remarks

- A subject file may be connected 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 are passed to each of the connected files.
- For more information, see the NOS/VE System Usage manual.

Examples

The following example connects subject file $ECHO to file ECHO_FILE.

```
/crefc $echo echo_file
```

This connection results in each subsequent command being echoed to file ECHO_FILE after it is interpreted. This is very useful when debugging SCL procedures since the commands within a procedure are not written to the log. Commands written to $ECHO are preceded by an identifier that indicates how the command was processed.

The following procedure is then created.

```
/colt new_proc
ct? proc new_proc
ct? display_catalog $user
ct? display_log 1
ct? display_value 'Test Complete.'
ct? procend
ct? **
```

The following output results when the procedure is executed.

```
/new_proc
CATALOG: CATALOG_1
CATALOG: CATALOG_2
   FILE: DATA_FILE_1
   FILE: EPILOG
   FILE: PROLOG
13:38:30.063.CI.colt new_proc
13:39:25.264.CI.new_proc
Test Complete.
```

The last line of the job log (Test Complete) is displayed to show that only the procedure call itself is written to the log.

```
/disl 1
13:39:25.264.CI.new_proc
13:39:40.552.CI.disl 1
```

However, file ECHO_FILE contains a list of each command executed within the procedure, as follows:

```
/delfc $echo echo_file
/copy_file echo_file
CI colt new_proc
CI new_proc
CI proc new_proc
CI display_catalog $user
CI display_log 1
CI display_value 'Test Complete.'
CI procend
CI disl 1
CI delfc $echo echo_file
```

CREATE_FILE_PERMIT Command

Purpose Establishes or modifies an access control entry for a specific file.

Format **CREATE_FILE_PERMIT** or
CREFP
FILE=file
GROUP=keyword
FAMILY_NAME=name
USER=name
ACCOUNT=name

PROJECT=name
ACCESS_MODES=list of keyword
SHARE_MODES=list of keyword
APPLICATION_INFORMATION=string
STATUS=status variable

Parameters

FILE or **F**

Specifies the permanent file for which the access control entry is being established. It determines the catalog in which the file is registered and the file name. This parameter is required.

GROUP or *G*

Specifies if the permit entry is for a specific user or group of users. The selections are:

PUBLIC

The permit entry applies to all users regardless of family, account, project, or user identifications.

FAMILY

The permit entry applies to all users in the specified family.

ACCOUNT

The permit entry applies to all users associated with the specified family and account identifications.

PROJECT

The permit entry applies to all users associated with the specified family, account, and project identifications.

USER

The permit entry applies to the user identified by the specified family and user identifications.

USER_ACCOUNT

The permit entry applies to the user identified by the specified family, account, and user identifications.

MEMBER

The permit entry applies to the user identified by the specified family, account, project, and user identifications.

Omission causes USER to be used.

FAMILY_NAME or *FN*

Specifies the family name to be permitted access. Omission causes the family name associated with the requesting job to be used if the GROUP selection indicates this parameter is applicable. If the GROUP selection indicates this parameter is not applicable and this parameter is specified, an abnormal status is returned.

USER or *U*

Specifies the user name to be permitted access. Omission causes the user name associated with the requesting job to be used if the GROUP selection indicates this parameter is applicable. If the GROUP selection indicates this parameter is not applicable and this parameter is specified, an abnormal status is returned.

ACCOUNT or *A*

Specifies the account to be permitted access. Omission causes the account associated with the requesting job to be used if the GROUP selection indicates this parameter is applicable. If the GROUP selection indicates this parameter is not applicable and this parameter is specified, an abnormal status is returned.

PROJECT or *P*

Specifies the project to be permitted access. Omission causes the project associated with the requesting job to be used if the GROUP selection indicates this parameter is applicable. If the GROUP selection indicates this parameter is not applicable and this parameter is specified, an abnormal status is returned.

ACCESS_MODES or *ACCESS_MODE* or *AM*

Specifies how the file may be used by the group specified. The access modes are:

READ

Access is permitted to read the file.

APPEND

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

MODIFY

Access is permitted to alter data within the existing file.

SHORTEN

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

WRITE

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

EXECUTE

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

ALL

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

CONTROL

Access is permitted to delete a cycle, to change its file identity (file name, cycle number, password, log selection, retention, and charge attributes) and to change its file attributes.

CYCLE

Access is permitted to add new file cycles.

NONE

Access is specifically prohibited.

Omission causes READ and EXECUTE to be used.

SHARE_MODES or *SHARE_MODE* or *SM*

Specifies how the file must be shared, as a minimum, by the group specified. The share modes are:

READ

Permitted jobs are required to share the file for READ access.

APPEND

Permitted jobs are required to share the file for APPEND access.

MODIFY

Permitted jobs are required to share the file for MODIFY access.

SHORTEN

Permitted jobs are required to share the file for SHORTEN access.

WRITE

Permitted jobs are required to share the file for SHORTEN, MODIFY, and APPEND access.

EXECUTE

Permitted jobs are required to share the file for EXECUTE access.

ALL

Permitted jobs are required to share the file for READ, APPEND, MODIFY, SHORTEN, WRITE, and EXECUTE access.

NONE

No sharing requirements are imposed on permitted jobs. When attaching the file, permitted jobs may select exclusive access or any of the READ, APPEND, MODIFY, SHORTEN, WRITE, or EXECUTE share modes.

Omission causes the share mode to be determined by the value specified on the ACCESS_MODE parameter of this command. If access mode includes APPEND, SHORTEN,

MODIFY, or WRITE, the share requirement chosen is NONE. Otherwise, the share requirements chosen are READ and EXECUTE.

APPLICATION_INFORMATION or *AI*

Specifies a string to be saved in the permit entry. An application can use this information for additional access control. A string of up to 31 characters can be specified. Omission causes a string of spaces to be used.

Remarks

- Access control also can be established at a catalog level that controls access to all files registered relative to the catalog.

  Refer to the CREATE_CATALOG_PERMIT command for more information.

- This request allows the file owner to specify who can use a file with what modes of access. This access control applies to all cycles of a file.

- Permission can be established for specific users or groups of users associated with a specific family, account, and project.

- If an access control entry already exists for the specified user or group of users, then the specified access modes, share modes, and application information replace the selections currently defined.

- For more information, see the NOS/VE System Usage manual.

Examples

Following is an example of creating a file permit and using the DISPLAY_CATALOG_ENTRY command to display the file permit.

```
/create_file_permit $user.catalog_1.data_file_1 ..
../group=user user=krj am=read sm=(read,execute)
/disce $user.catalog_1.data_file_1 do=permits
   PERMIT_GROUP: USER
     FAMILY: NVE, USER: KRJ
     PERMITS: READ
     SHARE: READ, EXECUTE
     APPLICATION_INFORMATION:
```

The file permit is created for file DATA_FILE_1 in subcatalog CATALOG_1 in the master catalog. The GROUP parameter specifies that the permission applies to a user and the USER parameter identifies the user as KRJ. With this file permit, user KRJ is given read access to the file, which can concurrently be assigned to another user in read or execute modes.

In the following example, a similar file permit is created.

```
/crefp $user.catalog_1.data_file_1 group=user user=dlh ..
../am=write sm=none
/disce $user.catalog_1.data_file_1 do=permits
   PERMIT_GROUP: USER
     FAMILY: NVE, USER: DLH
     PERMITS: SHORTEN, APPEND, MODIFY
     SHARE: NONE
     APPLICATION_INFORMATION:
   PERMIT_GROUP: USER
     FAMILY: NVE, USER: KRJ
     PERMITS: READ
     SHARE: READ, EXECUTE
     APPLICATION_INFORMATION:
```

The preceding example gives user DLH append, modify, and shorten access to the same file and specifies that no share mode restrictions are required.

CREATE_IBM_REQUEST Command

Purpose Creates a NOS/VE temporary file to be associated with an IBM tape file. Future references to the IBM tape file are through the NOS/VE temporary file. The IBM tape file must be on an ANSI labeled tape.

Format

CREATE_IBM_REQUEST or
CREIR
FILE=file
EXTERNAL_VSN=list of string
RECORDED_VSN=list of string
FILE_SET_POSITION=keyword
FILE_IDENTIFIER=string
FILE_SEQUENCE_NUMBER=integer
GENERATION_NUMBER=integer
CHARACTER_CONVERSION=boolean
STATUS=status variable

Parameters FILE or F

Specifies the name of a NOS/VE temporary file to be associated with an IBM tape file. This parameter is required.

EXTERNAL_VSN or *EVSN* or *VSN*

Gives the external identification of the tape volume(s) containing the IBM tape file. Each parameter value is a string 1 to 6 characters long.

If you specify more than one external volume serial number (VSN), the volumes are requested in the order specified in the parameter list. If you omit the EXTERNAL_VSN parameter, the system uses the RECORDED_VSN parameter in its place.

You must specify either the EXTERNAL_VSN parameter or the RECORDED_VSN parameter. Otherwise, a fatal error results.

RECORDED_VSN or *RVSN*

Gives the VSN recorded internally on the ANSI VOL1 label on the tape volume(s) holding the IBM tape file. Each parameter value is a string 1 to 6 characters long. File processing uses the RECORDED_VSN parameter to locate and verify the correct volume.

If you specify more than one recorded VSN, the volumes are located and verified in the order specified in the parameter list. If you omit the RECORDED_VSN parameter, the system uses the EXTERNAL_VSN parameter to verify the VSN recorded internally on the ANSI VOL1 label.

If you specify both the EXTERNAL_VSN and RECORDED_VSN parameters, they are matched; the first external VSN with the first recorded VSN, the second external VSN with the second recorded VSN, and so on. For each such pair, NOS/VE uses the external VSN parameter to direct the system operator to mount the tape with that external VSN. NOS/VE uses the recorded VSN value to verify the VSN recorded internally on the ANSI VOL1 label on that tape.

If there is an EXTERNAL_VSN parameter with no matching RECORDED_VSN parameter, NOS/VE uses the EXTERNAL_VSN parameter to direct the system

operator. NOS/VE also uses the EXTERNAL_VSN parameter to verify the VSN recorded internally on the ANSI VOL1 tape label.

If there is a RECORDED_VSN parameter with no matching EXTERNAL_VSN parameter, NOS/VE uses the RECORDED_VSN parameter to direct the system operator. NOS/VE also uses the RECORDED_VSN parameter to verify the VSN recorded internally on the ANSI VOL1 tape label.

FILE_SET_POSITION or *FSP*

Specifies the position of the IBM tape file to be read. If you omit this parameter, the NEXT_FILE position is assumed. The parameter can have any of the following values:

BEGINNING_OF_SET or BOS

Specifies that the first tape file on the file set is to be read.

CURRENT_FILE or CF

Specifies that the current tape file is to be read. That is, the last tape file accessed will be accessed again. If the tape is positioned at the beginning of the first volume, the first tape file will be read.

FILE_IDENTIFIER_POSITION or FIP

Specifies that the tape file identified by the FILE_IDENTIFIER and GENERATION_NUMBER parameters is to be read.

FILE_SEQUENCE_POSITION or FSP

Specifies that the tape file identified by the FILE_SEQUENCE_NUMBER parameter is to be read.

NEXT_FILE or NF

Specifies that the tape file following the last accessed tape file will be read. If the tape is positioned at the beginning of the first volume, the first tape file will be read.

FILE_IDENTIFIER or *FI*

Specifies a file identifier as a string of 1 to 17 characters. Each tape file on a multifile set has a unique file identifier. If you specify the FILE_IDENTIFIER_POSITION value for the FILE_SET_POSITION parameter, the FILE_IDENTIFIER parameter is required; otherwise, its value is ignored.

FILE_SEQUENCE_NUMBER or *FSN*

Specifies the numeric position of a tape file on a multifile set. The position is an unsigned integer in the range 1 through 9999. If you specify the FILE_SEQUENCE_POSITION value for the FILE_SET_POSITION parameter, the FILE_SEQUENCE_NUMBER parameter is required; otherwise, its value is ignored.

GENERATION_NUMBER or *GN*

Identifies a specific revision of the tape file named by the FILE_IDENTIFIER parameter. The revision shows as an unsigned integer in the range 1 through 9999. If the FILE_SET_POSITION parameter has the FILE_IDENTIFIER_POSITION value, and the GENERATION_NUMBER parameter is omitted, then the GENERATION_NUMBER parameter value is set to one.

CHARACTER_CONVERSION or *CC*

Specifies with a boolean value whether or not the tape file data is to be converted to or from its character set. If you omit the CHARACTER_CONVERSION parameter, FALSE is assumed to be its value.

Of the tape file migration methods, only FMU automatically does character conversion in addition to any conversion specified by the CHARACTER_CONVERSION parameter value.

To obtain a properly migrated tape file, you usually want to set the CHARACTER_CONVERSION parameter to FALSE if you use FMU to migrate. Otherwise you convert your tape file data twice. Set this parameter to TRUE if you use any other tape file migration method. Otherwise, you do not convert your tape file data at all.

Remarks

- You must enter a CREATE_IBM_REQUEST command before the FMU command.
- For more information, see the Migration From IBM to NOS/VE Tutorial/Usage manual.

Examples

The following command associates temporary file FILE_FTN with a typical FORTRAN formatted tape file:

```
/create_ibm_request  file=file_ftn ..
../external_vsn='osar' ..
../recorded_vsn='audi' ..
../file_set_position=file_identifier_position ..
../file_identifier='cm_pn' ..
../generation_number=23 ..
../character_conversion=true
```

The tape file has the following characteristics:

- Volume with external vsn, OSAR; recorded vsn, AUDI.
- Found from its file identifier, CM_PN, and generation number, 23.
- Tape file data to be converted from its character set.

CREATE_INTERSTATE_CONNECTION Command

Purpose

Establishes a NOS batch control point on a dual state system.

Format

CREATE_INTERSTATE_CONNECTION or **CREIC**

PARTNER_JOB_CARD=string
STATUS=status variable

Parameters

PARTNER_JOB_CARD or *PJC*

Specifies the job statement parameters to be used for the NOS batch job. The parameter syntax must conform to NOS job statement rules.

Omission causes the NOS default job statement parameters to be used (an infinite time limit and no other parameters specified).

Remarks

- After you enter a CREATE_INTERSTATE_CONNECTION command, prompts are issued until you enter QUIT (QUI) or DELETE_INTERSTATE_CONNECTION (DELIC).
- While the interstate connection is open, you can enter any NOS/VE command (except another CREIC command). You can enter NOS commands to be executed on the NOS side of the dual state system through the EXECUTE_INTERSTATE_COMMAND command. The CREIC command is generally used in conjunction with the File Management Utility to migrate files between NOS and NOS/VE.
- For more information, see the NOS/VE Advanced File Management Usage manual.

Examples

The following commands create an interstate connection, execute NOS commands (ATTACH, DEFINE, and COPY), and close the connection. FA is the CREATE_INTERSTATE_CONNECTION prompt for user input.

```
/create_interstate_connection partner_job_card=..
../'myjob,,64.'
FA/execute_interstate_command command='attach,oldfil.'
FA/execute_interstate_command command='define,newfil.'
FA/execute_interstate_command command=..
FA../'copy,oldfil,newfil.'
FA/delete_interstate_connection
/
```

CREATE_KEYED_FILE Command

Purpose

Begins a CREATE_KEYED_FILE utility session.

Format

CREATE_KEYED_FILE or
CREATE_KEYED_FILES or
CREKF
OUTPUT=file
STATUS=status variable

Parameters

OUTPUT or O

File path of the keyed file to be created. The keyed-file attributes must already be specified by SET_FILE_ATTRIBUTES commands.

This parameter is required.

The minimum attributes that must be defined are KEY_LENGTH and MAXIMUM_RECORD_LENGTH. If the FILE_ORGANIZATION is omitted, Create_Keyed_File creates an indexed-sequential file.

Remarks

- The command utility prompt is:

  ```
  crekf/
  ```

  In response to the crekf/ prompt, you can enter SCL commands and any of these subcommands:

  ADD_RECORDS
  REPLACE_RECORDS
  COMBINE_RECORDS
  EXTRACT_RECORDS
  DISPLAY_RECORDS
  DELETE_RECORDS
  CREATE_NESTED_FILE
  SELECT_NESTED_FILE
  DELETE_NESTED_FILE
  DISPLAY_NESTED_FILE
  CREATE_ALTERNATE_INDEXES
  HELP
  QUIT

- The new keyed file is created with one nested file, named $MAIN_FILE. It is the initially selected nested file and all subcommands apply to it until a CREATE_NESTED_FILE or SELECT_NESTED_FILE subcommand selects another nested file.

- If any nested file in the new keyed file uses a user-defined collation table, hashing procedure, or compression procedure, the object library containing the compiled table or procedure must be in the program library list before the Create_Keyed_File session begins.

  To add one or more object libraries to the program library list, use the ADD_LIBRARIES parameter on a SET_PROGRAM_ATTRIBUTES command. For example:

```
set_program_attributes, add_library=$user.hash_library
```

- If you specify DIRECT_ACCESS as the FILE_ORGANIZATION attribute on the SET_FILE_ATTRIBUTES command, but omit the INITIAL_HOME_BLOCK_COUNT attribute, CREATE_KEYED_FILE prompts you for calculation of the INITIAL_HOME_BLOCK_COUNT.

- For more information, see the NOS/VE Advanced File Management Usage manual.

Examples This CREATE_KEYED_FILE example defines the file $USER.INDEXED SEQUENTIAL_FILE with the SET_FILE_ATTRIBUTES command and then creates it.

```
/set_file_attributes, file=$user.indexed_sequential_file ..
../file_organization=indexed_sequential ..
../maximum_record_length=32, minimum_record_length=14 ..
../key_length=14
/create_keyed_file, output=$user.indexed_sequential_file
crekf/
```

CREATE_MANUAL Command

Purpose Reads a sequential (source) file containing directives and text; creates a segmented access (bound) online manual file which can be named in an EXPLAIN command MANUAL parameter.

Format

CREATE_MANUAL or
CREM
INPUT=file
OUTPUT=file
ERROR=file
LIST=file
LIST_OPTIONS=keyword
STATUS=status variable

Parameters

INPUT or I

File reference for the source file containing online manual text and directives. This parameter is required.

OUTPUT or O

File reference for the bound file containing the finished online manual. The default file is $LOCAL.MANUAL.

ERROR or *E*

File reference for the listing file to receive error messages. The default file is $LOCAL.$ERRORS (the terminal).

LIST or *L*

File reference for the listing file to receive a cross-reference listing of screen names and indexed subjects. If you do not specify the LIST_OP parameter, no cross-reference listing is produced. The default file is $LIST.

LIST_OPTIONS or *LO*

Specifies whether the LIST file should contain information. Possible values:

R

Produce the listing

NONE

Do not produce the listing (default)

Remarks For more information, see the CYBER Online Text manual.

Examples The following command creates the manual called CONTEXT from a source file called CONTEXT_SOURCE. An error file called CONTEXT_ERRORS and a cross-reference file called CONTEXT_CROSS_REFERENCE are produced.

```
/create_manual input=$user.context_source  ..
../output=$user.context  ..
../error_list=$user.context_errors ..
../list=$user.context_cross_reference ..
../list_op=r
```

CREATE_OBJECT_LIBRARY Command

Purpose Begins a CREATE_OBJECT_LIBRARY utility session. The utility produces an object library or an object file and allows post-compilation manipulation of object or load modules. It can also produce a text version of certain kinds of modules on an object library.

Format **CREATE_OBJECT_LIBRARY** or
CREOL
STATUS=status variable

Remarks

- The following files can be created by the GENERATE_LIBRARY subcommand of this utility. The utility issues a warning and does not process input files whose file attributes do not conform to the attributes listed in the right-hand column. The utility sets the accompanying file attributes listed for output files it creates. You can override attributes of a file with the SCL SET_FILE_ATTRIBUTE command.

File Created	Attributes Given to the File
Object library	FILE_CONTENT=OBJECT FILE_STRUCTURE=LIBRARY
Object file	FILE_CONTENT=OBJECT FILE_STRUCTURE=DATA
SCL procedure file	FILE_CONTENT=LEGIBLE FILE_PROCESSOR=SCL FILE_STRUCTURE=DATA
File containing the subcommands that define message modules	FILE_CONTENT=LEGIBLE FILE_PROCESSOR=SCL FILE_STRUCTURE=DATA

- The CREOL session ends when you enter the QUIT subcommand.
- For more information, see the NOS/VE Object Code Management manual.

Examples The following is a sequence that removes an object library from the command list, creates a new version of the object library from the modules on file $LOCAL.LGO, then adds the object library to the command list.

```
/delete_command_list_entry entry=$local.my_commands
/create_object_library
COL/add_module $local.lgo
COL/generate_library $local.my_commands
COL/quit
/create_command_list_entry entry=$local.my_commands
/
```

CREATE_PROGRAM_PROFILE Command

Purpose Generates a program profile.

Format

CREATE_PROGRAM_PROFILE or
CREPP
TARGET_TEXT=file
FILE=list of file
PARAMETER=string
LIBRARY=list of file
MODULE=list of any
STARTING_PROCEDURE=any
PROFILE_ORDER=keyword
PROGRAM_UNIT_CLASS=keyword
NUMBER=integer or *keyword*
OUTPUT=file
STATUS=status variable

Parameters

TARGET_TEXT or TT

File containing the modules to be measured. This parameter is required.

FILE or *FILES* or *F*

Object list for the program. Each module in the specified object files and object libraries is unconditionally included in the program. The list must include the target text file. If FILE is omitted, the object list for the program consists of only the file specified on the TARGET_TEXT parameter.

PARAMETER or *P*

Parameter list passed to the program.

LIBRARY or *LIBRARIES* or *L*

List of object libraries added to the program library list. These object libraries are searched before any libraries in the job library list. The libraries are searched in the order listed.

MODULE or *MODULES* or *M*

Module list.

You use a string value for a module whose name is not an SCL name.

Each module is unconditionally loaded from the object libraries in the program library list.

STARTING_PROCEDURE or *SP*

Name of the entry point where execution begins.

You use a string value for an entry point whose name is not an SCL name.

If STARTING_PROCEDURE is omitted, the last transfer symbol encountered during loading is used.

PROFILE_ORDER or *PO*

Order in which the program profile is displayed. Options are:

TIME (T)

By percentage of time spent executing ordered greatest to least.

PROGRAM_UNIT (PU)

By program unit name ordered alphabetically.

MODULE_PROGRAM_UNIT (MPU)

By module name ordered alphabetically.

If PROFILE_ORDER is omitted, TIME is used.

PROGRAM_UNIT_CLASS or *PUC*

Class of program units whose statistics are displayed. Options are:

ALL

All program units measured, both local and remote.

LOCAL

Only program units that are part of the target text.

REMOTE

Only program units that are called by target text program units, but are not part of the target text. These program units provide the remote block statistics in the program profile.

If PROGRAM_UNIT_CLASS is omitted, ALL is used.

NUMBER or *N*

Number of program unit statistics displayed. The statistics are sorted as specified by the PROFILE_ORDER parameter and then displayed in order until the specified number of statistics have been displayed. If NUMBER is omitted, the entire program profile is displayed.

OUTPUT or *O*

File to which the display is written. This file can be positioned. If OUTPUT is omitted, file $OUTPUT is used.

Remarks

- The CREATE_PROGRAM_PROFILE command executes the program described on the command and accumulates the execution time of each of the program units. It then generates a program profile listing the execution time statistics. The command does not generate a connectivity matrix or a restructuring procedure for the program.
- For more information, see the NOS/VE Object Code Management manual.

Examples

The following command executes the program on file LGO and saves the program profile on file PROFILE_LIST.

```
/create_program_profile lgo output=profile_list
```

CREATE_REMOTE_VALIDATION Command

Purpose Provides remote system validation information for implicit access to remote NOS/VE files or for the MANAGE_REMOTE_FILES command.

Format

CREATE_REMOTE_VALIDATION or
CRERV
LOCATION=name
VALIDATION=list of string
STATUS=status variable

Parameters

LOCATION or **L**

Specifies the name of the remote location to be accessed. For implicit remote file access, this is the family name of the remote NOS/VE system that you want to access.

For explicit remote access through MANAGE_REMOTE_FILES command, this is a name associated with the remote system, such as a family name or a logical identifier. (Location names are determined by your network application administrator.)

This parameter is required.

VALIDATION or **V**

Specifies the lines of text used to validate access to the remote location. If the remote system is a NOS/VE system, the first (or only) string of this parameter must be a NOS/VE LOGIN command.

Refer to the Remote Host Facility Usage manual for information about validation commands required by non-NOS/VE systems.

This parameter is required.

Remarks

- You must enter this command before you can implicitly reference a remote file or catalog. This command is optional if you explicitly access a remote system using the MANAGE_REMOTE_FILE command. If you use CREATE_REMOTE_VALIDATION in this case, the MANAGE_REMOTE_FILES command sends the validation information so it precedes any other commands sent to that location.

- The validation information specified by this command remains in effect until the job terminates or until you enter a DELETE_REMOTE_VALIDATION command, which deletes the validation information.
- You can use this command to establish validation information for more than one remote location. You must enter a separate CREATE_REMOTE_VALIDATION command for each remote location.
- For more information, see the NOS/VE System Usage manual.

Examples The following example establishes access to remote family SKY on a remote system for user name MIKE_B, family name SKY, and password STARS:

```
/create_remote_validation location=sky ..
../validation='login user=mike_b ..
../fn=sky pw=stars'
```

CREATE_VARIABLE Command

Purpose Creates an SCL variable.

Format

CREATE_VARIABLE or
CREATE_VARIABLES or
CREV
NAMES=list of name
KIND=list of any
DIMENSION=range of integer
VALUE=any
SCOPE=name
STATUS=status variable

Parameters

NAMES or **NAME** or **N**

Specifies the name(s) of the variable(s) to be created. This parameter is required.

KIND or *K*

Specifies the type or kind of variable to be created. The following are valid keywords for the KIND parameter:

INTEGER

Creates an integer variable.

BOOLEAN

Creates a boolean variable.

STRING

Creates an ASCII character string variable.

STATUS

Creates a status variable.

If you do not specify a keyword value element for this parameter, INTEGER is assumed.

The integer element of the KIND parameter specifies the maximum length for a string variable; it cannot be specified for other variable kinds. If the integer element is omitted for a string variable, the maximum length is assumed to be 256.

DIMENSION or *D*

Specifies the lower and upper bounds of an array. This parameter is used only when the variable being created is an array. You can specify a range from -2,147,483,647 to 2,147,483,647. If you omit the DIMENSION parameter, the system assumes that the variable is not an array (that is, a dimension of 1..1 is assumed).

VALUE or *V*

Specifies the initial value of the variable to be created. 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.

- You cannot use the VALUE parameter to assign values to specific fields of a variable of type status. You must either use the system default (see below) or the $STATUS function.

If you omit the VALUE parameter, the following default values are assigned:

Kind of Variable	Default Value
Integer	0
String of zero	" (null string)
Boolean	FALSE
Status	NORMAL field is TRUE IDENTIFIER field is undefined CONDITION field is undefined TEXT field is undefined

SCOPE or *S*

Specifies the scope of the variable. The scope of a variable defines the blocks in which the variable can be accessed. The following are valid entries for the SCOPE parameter:

LOCAL

Causes the variables to be local to the current block. These variables cannot be referenced by other blocks. This is the default.

XDCL

Causes the variables to be 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. A variable's KIND and DIMENSION attributes can be returned using the $VARIABLE function.

XREF

States that the variables are externally referenced (XREF). A variable with this scope must have been created in an outer block with the scope XDCL and with KIND and DIMENSION parameters identical to

those supplied by this command. A variable's KIND and DIMENSION parameters can be returned using the $VARIABLE function.

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.

Remarks

- If you specify a scope of XREF, any value you specify for the VALUE parameter is ignored.
- For more information, see the NOS/VE System Usage manual.

Examples

- The following example creates a variable named GLOBAL_STATUS, of kind STATUS, with a scope of JOB:

```
/create_variable global_status kind=status ..
../scope=job
```

  The GLOBAL_STATUS variable can be referenced from any block from which it is externally referenced (via the SCOPE=XREF parameter).

- The next example creates a variable named DONE of kind 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 name=done kind=boolean ..
../value=false scope=xdcl
```

CREATE_VAX_REQUEST Command

Purpose Creates a NOS/VE temporary file to be associated with a VAX tape file. Future references to the VAX tape file are through the NOS/VE temporary file. The VAX tape file must be on an ANSI labeled tape.

Format

CREATE_VAX_REQUEST or
CREVR
FILE=file
EXTERNAL_VSN=list of string
RECORDED_VSN=list of string
FILE_SET_POSITION=keyword
FILE_IDENTIFIER=string
FILE_SEQUENCE_NUMBER=integer
GENERATION_NUMBER=integer
STATUS=status variable

Parameters

FILE or F

Specifies the name of a NOS/VE temporary file to be associated with a VAX tape file. This parameter is required.

EXTERNAL_VSN or *EVSN* or *VSN*

Gives the external identification of the tape volume(s) containing the VAX tape file. Each parameter value is a string 1 to 6 characters long.

If you specify more than one external volume serial number (VSN), the volumes are requested in the order specified in the parameter list. If you omit the EXTERNAL_VSN parameter, the system uses the RECORDED_VSN parameter in its stead.

You must specify either the EXTERNAL_VSN parameter or the RECORDED_VSN parameter. Otherwise, a fatal error results.

RECORDED_VSN or *RVSN*

Gives the VSN recorded internally on the ANSI VOL1 label on the tape volume(s) holding the VAX tape file. Each parameter value is a string 1 to 6 characters long. File processing uses the RECORDED_VSN parameter to locate and verify the correct volume.

If you specify more than one recorded VSN, the volumes are located and verified in the order specified in the parameter list. If you omit the RECORDED_VSN parameter, the EXTERNAL_VSN parameter is used to verify the VSN recorded internally on the ANSI VOL1 label.

If you specify both the EXTERNAL_VSN and RECORDED_VSN parameters, they are matched; the first external VSN with the first recorded VSN, the second external VSN with the second recorded VSN, and so on. For each such pair, NOS/VE uses the external VSN parameter to direct the system operator to mount the tape with that external VSN. NOS/VE uses the recorded VSN value to verify the VSN recorded internally on the ANSI VOL1 label on that tape.

If there is an EXTERNAL_VSN parameter with no matching RECORDED_VSN parameter, NOS/VE uses the EXTERNAL_VSN parameter to direct the system operator. NOS/VE also uses the EXTERNAL_VSN parameter to verify the VSN recorded internally on the ANSI VOL1 tape label.

If there is a RECORDED_VSN parameter with no matching EXTERNAL_VSN parameter, NOS/VE uses the RECORDED_VSN parameter to direct the system operator. NOS/VE also uses the RECORDED_VSN parameter to verify the VSN recorded internally on the ANSI VOL1 tape label.

FILE_SET_POSITION or *FSP*

Specifies the position of the VAX tape file to be read. If you omit this parameter, the NEXT_FILE position is assumed. The parameter can have any of the following values:

BEGINNING_OF_SET or BOS

Specifies that the first tape file on the file set is to be read.

CURRENT_FILE or CF

Specifies that the current tape file is to be read. That is, the last tape file accessed will be accessed again. If the tape is positioned at the beginning of the first volume, the first tape file will be read.

FILE_IDENTIFIER_POSITION or FIP

Specifies that the tape file identified by the FILE_IDENTIFIER and GENERATION_NUMBER parameters is to be read.

FILE_SEQUENCE_POSITION or FSP

Specifies that the tape file identified by the FILE_SEQUENCE_NUMBER parameter is to be read.

NEXT_FILE or NF

Specifies that the tape file following the last accessed tape file will be read. If the tape is positioned at the beginning of the first volume, the first tape file will be read.

FILE_IDENTIFIER or *FI*

Specifies a file identifier as a string of 1 to 17 characters. Each tape file on a multifile set has a unique file identifier. If you specify the FILE_IDENTIFIER_POSITION value for the FILE_SET_POSITION parameter, the FILE_IDENTIFIER parameter is required; otherwise, its value is ignored.

FILE_SEQUENCE_NUMBER or *FSN*

Specifies the numeric position of a tape file on a multifile set. The position is an unsigned integer in the range 1 through 9999. If you specify the FILE_SEQUENCE_POSITION value for the FILE_SET_POSITION parameter, the FILE_SEQUENCE_NUMBER parameter is required; otherwise, its value is ignored.

GENERATION_NUMBER or *GN*

Identifies a specific revision of the tape file named by the FILE_IDENTIFIER parameter. The revision shows as an unsigned integer in the range 1 through 9999. If the FILE_SET_POSITION parameter has the FILE_

IDENTIFIER_POSITION value, and the GENERATION_NUMBER parameter is omitted, then the GENERATION_NUMBER parameter value is set to one.

Remarks

- For more information, see the Migration From VAX/VMS to NOS/VE Tutorial/Usage manual.
- You must enter a CREATE_VAX_REQUEST command before the FMU command.

Examples

The following command associates temporary file FILE_FTN with a typical FORTRAN formatted tape file:

```
/create_vax_request  file=file_ftn ..
../external_vsn='carp' ..
../recorded_vsn='fish' ..
../file_set_position=file_identifier_position ..
../file_identifier='lm_no_goldfish' ..
../generation_number=2
```

The tape file has the following characteristics:

- Volume with external vsn, CARP; recorded vsn, FISH.
- Found from its file identifier, LM_NO_GOLDFISH, and generation number, 2.

CYBIL Command

Purpose

Calls the compiler, specifies the files to be used for input and output, and indicates the type of output to be produced.

Format

CYBIL
INPUT=file
BINARY=file
LIST=file
INPUT_SOURCE_MAP=file
LIST_OPTIONS=list of keyword
DEBUG_AIDS=list of keyword
ERROR_LEVEL=keyword
OPTIMIZATION_LEVEL=keyword
CA=integer

PAD=integer
RUNTIME_CHECKS=list of keyword
OPTIMIZATION_OPTIONS=list of keyword
KEYPOINT_GENERATION=boolean
STATUS=status variable

Parameters *INPUT* or *I*

Specifies the file that contains the source text to be read. Source input ends when an end-of-partition or an end-of-information is encountered on the source input file. If omitted, $INPUT is assumed.

BINARY or *B* or *BINARY_OBJECT*

Specifies the file on which object code is to be written. You can specify a file position as part of the file name. If $NULL is specified, the compiler performs a syntactic and semantic scan of the program but does not generate object code. If omitted, $LOCAL.LGO is assumed.

LIST or *L*

Specifies the file on which the compilation listing is to be written. If $NULL is specified, all compile-time output is discarded. If omitted, $LIST is assumed.

INPUT_SOURCE_MAP or *ISM*

Reserved.

LIST_OPTIONS or *LO*

Specifies a combination of the following list options. If NONE is specified, no list options are selected. If omitted, option S (list the source input file) is assumed.

A

Produces an attribute list of source input block structure and relative stack. The attribute listing is produced following the source listing on the file specified by the LIST parameter or, if the LIST parameter is omitted, on file $LIST.

F

Produces a full listing. In effect, this option selects options A, S, and R.

O

Lists compiler-generated object code. When selected, this listing includes an assembly-like listing of the generated object code. This option has no effect if the BINARY_OBJECT parameter is set to $NULL.

R

Produces a symbolic cross-reference listing showing the location of a program entity definition and its use within a program.

RA

Produces a symbolic cross-reference listing of all program entitites whether referenced or not.

S

Lists the source input file.

X

Used in conjunction with the compile-time directive LISTEXT so that listings can be externally controlled using the CYBIL command. The LISTEXT toggle must be ON.

DEBUG_AIDS or *DA*

Specifies a combination of the following Debug options. If omitted, NONE (no debug options) is assumed. Options are:

ALL

Selects debug options DS and DT.

DS

Compiles all debugging statements. A debugging statement is a statement in the source text that is ignored unless this option is specified. These statements are enclosed by the compile-time directives COMPILE and NOCOMPILE. The symbol table and the line table for interactive debugging are also generated.

DT

Generates debug tables (that is, the symbol table and line table) as part of the object code. These tables are used by the Debug utility.

NONE

No debug options are selected.

ERROR_LEVEL or *EL*

Specifies one of the following error list options. If omitted, W (list warning and fatal diagnostics) is assumed.

F

Lists fatal diagnostics. If selected, only fatal diagnostics are listed.

W

Lists warning (informative) diagnostics as well as fatal diagnostics.

OPTIMIZATION_LEVEL or *OL* or *OPTIMIZATION* or *OPT*

Specifies one of the following optimization options. If omitted, LOW is assumed.

DEBUG

Object code is stylized to facilitate debugging. Stylized code contains a separate packet of instructions for each executable source statement; it carries no variable values across statement boundaries in registers, and it notifies Debug each time the beginning of a statement or procedure is reached.

LOW

Provides for keeping constant values in registers.

HIGH

Provides for keeping local variables in registers, passing parameters to local procedures in registers, and eliminating redundant memory references, common subexpressions, and jumps to jumps.

CA

Reserved.

PAD

Generates the specified number of no-op (no operation) instructions between instructions that actually perform operations. If omitted, 0 (zero) is assumed; no-op instructions are not generated.

RUNTIME_CHECKS or *RC*

Specifies a combination of the following run-time checking options. If omitted, NONE (no run-time checks) is assumed.

ALL

Selects run-time checking options N, R, and S.

N

Produces compiler-generated code that checks for a NIL value when a reference is made to the object of a pointer.

NONE

No run-time checks are produced.

R

Produces compiler-generated code to check ranges. Range checking code is generated for assignment to integer subranges, ordinal subranges, and character variables. All CASE statements are checked to ensure that the selection expression corresponds to one of the variant values specified if no ELSE clause is provided. All references to substrings are verified. If an offset (variable pointer) is specified on a RESET statement, it is checked to ensure that it is valid for the specified sequence.

S

Produces compiler-generated code to test the subscripting of arrays.

OPTIMIZATION_OPTIONS or *OO*

Controls the optimizations applied to code generation. The nature of the generated code (including optimizations) is primarily determined by the OPTIMIZATION_LEVEL

parameter. The OPTIMIZATION_OPTIONS parameter, however, controls certain aspects of optimization, as follows. If this parameter is omitted, NONE is assumed.

INSTRUCTION_SCHEDULING or IS

To increase execution speed, instruction sequences are rearranged to take advantage of the vector architecture of the CYBER 990. [1] To select this option, you must also set the OPTIMIZATION_LEVEL parameter to HIGH.

NONE

Instruction scheduling does not take place.

KEYPOINT_GENERATION or *KG*

Boolean value that specifies whether or not object code keypoint instructions are generated for #KEYPOINT procedures in the source file.#KEYPOINT is a tool for analyzing program performance. For more information on #KEYPOINT, refer to chapter 7. To produce optimal object code, however, it is better to turn off #KEYPOINT generation. If this parameter is omitted, TRUE is assumed.

TRUE

Object code is generated for #KEYPOINT instructions. During execution, the #KEYPOINT object code is skipped (unless it is specifically enabled). Skipping the #KEYPOINT object code involves a certain amount of CPU overhead, which decreases program performance.

FALSE

No object code is generated for #KEYPOINT instructions. To improve program performance when you do not want keypoint processing, specify FALSE.

1. Instruction scheduling on other CYBER machines has little or no effect on execution speed.

Remarks

- If the compiler command specifies an option that differs from a directive, the latest occurrence of either the command or the directive takes precedence.
- For more information, see the CYBIL Language Definition manual.

Examples

This command reads source code from a file named COMPILE, writes the compilation file on file LIST, and writes the object code on file BIN1. The listing includes source code, compiler-generated object code, and a symbolic cross-reference listing.

```
/cybil i=compile l=list b=bin1 lo=(o,r)
```

CYCLE Control Statement

Purpose

Causes execution of the next iteration, if any, of a repetitive statement.

Format

CYCLE
label
WHEN boolean expression

The following are valid forms of the CYCLE statement:

CYCLE
CYCLE *label*
CYCLE *WHEN boolean expression*
CYCLE *label WHEN boolean expression*

Parameters

label

Specifies a label associated with the enclosing repetitive statement to be cycled. If the label is omitted, the innermost repetitive statement is cycled.

boolean expression

Specifiies whether exiting from the designated enclosing statement should take place. If the expression is TRUE, cycling is performed. If the expression is FALSE, or if it is omitted, cycling is not performed.

Remarks

- In the case of a LOOP statement, the first statement of the LOOP statement list is given control.
- In the case of a WHILE or REPEAT statement, the boolean expression controlling the loop is evaluated.
- In the case of a FOR statement, the processing done at the FOR statement's FOREND clause is performed.
- The WHEN clause is evaluated at the point of reference rather than continually, as is done by the WHEN condition handler.
- For more information, see the NOS/VE System Usage manual.

Examples

In the block structure example on the following screen, if the first CYCLE is executed, the LOOP statement labeled BLOCK1 is cycled.

If the second CYCLE is executed, the LOOP statement labeled BLOCK1 is also cycled. Thus, both BLOCK2 and BLOCK1 are effectively cycled.

If the third CYCLE statement is executed, the LOOP statement labeled BLOCK2 is cycled and control remains in BLOCK2.

```
+-------+-->block1: loop
|       |       .
|       |       .
|       +------ cycle -          First CYCLE statement
|               .
|               .
|       +------>block2: loop
|       |           .
|       |           .
+------ | --------- cycle block1 Second CYCLE statement
        |           .
        |           .
        +---------- cycle        Third CYCLE statement
                    .
                loopend block2
                    .
            loopend block1
```

$DATE Function

Purpose Returns the current date as a string.

Format **$DATE**
(*keyword*)

Parameters *keyword*

Specifies the form in which the date is to be returned. The following keywords are valid:

DMY

Returns an 8-character date as shown in the following example:

```
28.03.87
```

MDY

Returns an 8-character date as shown in the following example:

```
03/28/87
```

MONTH

Returns the date as shown in the following example:

```
March 28, 1987
```

The length of the returned string is variable.

ISOD or ISO

Returns a 10-character date as shown in the following example:

```
1987-03-28
```

ISO is an acronym for the International Standards Organization.

ORDINAL

Returns a 7-character Julian date in the form of the year and number of day in the year. For example:

```
1987087
```

DEFAULT

Returns the default format of the date. This value is determined by your site.

If you omit this parameter, DEFAULT is used.

Remarks For further information about functions, see the NOS/VE System Usage manual.

Examples The following example returns the ordinal form of the current date:

```
/display_value 'The current date is:  '//$date..
../(ordinal)
The current date is:  1987087
```

DEBUG_PROGRAM Command

Remarks Reserved for site personnel, Control Data, or future use.

DEFINE_PRIMARY_TASK Command

Purpose Designates the requesting task as the primary task for the job. It is the primary task or its innermost, synchronous child task to which break conditions such as terminate break and pause break are sent.

This command can be used to designate an asynchronous task as the primary task.

When the task which issues this command terminates, its parent task becomes the primary task.

Format **DEFINE_PRIMARY_TASK** or
DEFPT
STATUS=status variable

Remarks For more information, see the NOS/VE System Usage manual.

DEFINE_TERMINAL Command

Purpose Compiles your terminal definition file, creating an object library of terminal definition modules.

Format

DEFINE_TERMINAL or
DEFT
INPUT=file
BINARY=file
LIST=file
STATUS=status variable

Parameters

INPUT or **I**

Specifies the terminal definition file you want to compile. Each input file can contain only one terminal definition. This parameter is required.

BINARY or *B*

Specifies the object library to contain the compiled terminal definition module. If the BINARY parameter is omitted, object library TERMINAL_DEFINITIONS under your working catalog is assumed.

LIST or *L*

Specifies the file you want to contain intermediate output from the compilation process (CYBIL code). Most users do not need to see this file. If omitted, $LIST is assumed. For interactive users, $LIST is not displayed at the terminal.

Remarks

- The first time you use the DEFINE_TERMINAL command it creates an object library that can be used by the Full Screen Editor and other screen-oriented applications.

  Thereafter, executing a DEFINE_TERMINAL command merges the new terminal definition with definitions you previously compiled (assuming you use the same object library name). Therefore, one library can contain all your compiled terminal definitions, even though each definition originates from its own file.

In the library, the terminal definition module is identified by the name you enter on the MODEL_NAME statement in the terminal definition file. It is the value for the MODEL parameter prefixed with the characters CSM$. If a module with the same name is already in the object library, the new module replaces the one in the library.

To delete modules from the object library, use the object library generator subcommand DELETE_MODULE.

You should keep your object library on a permanent file. By default, executing the DEFINE_TERMINAL command with your working catalog set to $USER merges your terminal definition into permanent file $USER.TERMINAL_DEFINITIONS.

- For more information, see the Terminal Definition Usage manual.

Examples

The following example shows how you would set up your own terminal definition for the Lear Siegler ADM5 terminal:

Set your working catalog to $USER and then copy sample deck CSM$SAMPLE from the source library $SYSTEM.CYBIL.OSF$PROGRAM_INTERFACE to your own file by entering:

```
sc/use_library base=$system.cybil.osf$program_interface
sc/extract_deck deck=csm$sample source=lear_siegler_adm5
sc/quit false
```

Then, having edited it, adding the needed information for the Lear Siegler ADM5 terminal (using MODEL_NAME=ADM5), you compile it:

```
/define_terminal input=lear_siegler_adm5
```

With the working catalog set to $USER, the terminal definition for the ADM5 terminal is merged into file $USER.TERMINAL_DEFINITIONS. The entry in the DISPLAY_OBJECT_LIBRARY listing of $USER.TERMINAL_DEFINITIONS is CSM$ADM5 and the model name is ADM5.

DELETE_CATALOG Command

Purpose Deletes a catalog and, optionally, its contents.

Format

DELETE_CATALOG or
DELC
CATALOG=file
DELETE_OPTION=keyword
STATUS=status variable

Parameters

CATALOG or **C**

Specifies the catalog to be deleted. This parameter is required.

DELETE_OPTION or *DO*

Specifies the parts of the catalog to delete. Options are as follows:

CATALOG_AND_CONTENTS (CAC)

The catalog and all of its contents (including any files and catalogs and their contents) is deleted.

CONTENTS_ONLY (CO)

All of a catalog's contents (including any files and catalogs and their contents) is deleted, but the catalog itself remains as an empty catalog.

ONLY_IF_EMPTY (OIE)

The catalog is deleted only if it is an empty catalog. This is the default.

Remarks

- This command releases all storage space associated with the deleted catalogs and mass storage files.
- Only the catalog owner or family administrator can delete a catalog.
- This command does not delete a master catalog. Only the family administrator can delete a master catalog.
- For more information, see the NOS/VE System Usage manual.

Examples The following example deletes subcatalog CATALOG_2 and all of its contents from the master catalog.

```
/delete_catalog catalog=$user.catalog_2 ..
../delete_options=catalog_and_contents
```

DELETE_CATALOG_PERMIT Command

Purpose Deletes an access control entry that was previously established for a catalog.

Format

DELETE_CATALOG_PERMIT or
DELCP
CATALOG=file
GROUP=keyword
FAMILY_NAME=name
USER=name
ACCOUNT=name
PROJECT=name
STATUS=status variable

Parameters

CATALOG or **C**

Specifies the catalog for which the access control entry is being deleted. This parameter is required.

GROUP or *G*

Specifies if the permit entry to be deleted applies to a specific user or group of users. The group selections are:

PUBLIC

All users regardless of family, account, project, or user identifications.

FAMILY

All users in the specified family.

ACCOUNT

All users associated with the specified family and account identifications.

PROJECT

All users associated with the specified family, account, and project identifications.

USER

The user identified by the specified family and user identifications.

USER_ACCOUNT

The user identified by the specified family, account, and user identifications.

MEMBER

The user identified by the specified family, account, project, and user identifications.

Omission causes USER to be used.

FAMILY_NAME or *FN*

Specifies the family name associated with the permit entry being deleted. Omission causes the family name associated with the requesting job to be used if the GROUP selection indicates this parameter is applicable. If the GROUP selection indicates this parameter is not applicable, an abnormal status is returned.

USER or *U*

Specifies the user name associated with the permit entry being deleted. Omission causes the user name associated with the requesting job to be used if the GROUP selection indicates this parameter is applicable. If the GROUP selection indicates this parameter is not applicable, an abnormal status is returned.

ACCOUNT or *A*

Specifies the account name associated with the permit entry being deleted. Omission causes the account associated with the requesting job to be used if the GROUP selection indicates this parameter is applicable. If the GROUP selection indicates this parameter is not applicable, an abnormal status is returned.

PROJECT or *P*

Specifies the project name associated with the permit entry being deleted. Omission causes the project associated with the requesting job to be used if the GROUP selection indicates this parameter is applicable. If the GROUP selection indicates this parameter is not applicable, an abnormal status is returned.

Remarks

- This request can be issued only by the owner of the catalog.
- For more information, see the NOS/VE System Usage manual.

Examples

The following example deletes the permission for user DLH to catalog CATALOG_1 in the master catalog, leaving only the permission for user KRJ.

```
/delete_catalog_permit $user.catalog_1 group=user ..
../user=dlh
/disc $user.catalog_1 permits
   PERMIT_GROUP: USER
     FAMILY: NVE, USER: KRJ
     PERMITS: SHORTEN, APPEND, MODIFY
     SHARE: NONE
     APPLICATION_INFORMATION:
```

Deletion of a catalog permit entry, however, does not affect individual file permit entries. That is, even though specific permission to CATALOG_1 for user DLH has been removed, user DLH can still access file DATA_FILE_0 for which specific file permission was previously established.

DELETE_COMMAND_LIST_ENTRY Command

Purpose

Deletes entries from the command list.

Format

DELETE_COMMAND_LIST_ENTRY or
DELETE_COMMAND_LIST_ENTRIES or
DELCLE
ENTRY=list of file or keyword
STATUS=status variable

Parameters

ENTRY or ENTRIES or E

Specifies the entries to be deleted from the command list. If this parameter is specified as $SYSTEM, the $SYSTEM command library is deleted from the command list. This parameter is required.

Remarks

- This command cannot be used when the command list search mode is EXCLUSIVE.
- If the command list search mode is RESTRICTED, the entry at the beginning of the command list cannot be deleted.
- For more information, see the NOS/VE System Usage manual.

DELETE_FILE Command

Purpose Deletes a file or file cycle.

Format

DELETE_FILE or
DELF
FILE=file
PASSWORD=name or *keyword*
STATUS=status variable

Parameters

FILE or **F**

Specifies the permanent file to be deleted. Omission of a cycle reference causes $LOW to be used. This parameter is required.

PASSWORD or *PW*

Specifies the file password. It must match the file password in the catalog entry; otherwise, the command is terminated and an error status is returned. Omission or specification of the keyword NONE causes no password to be used.

Remarks

- All storage space is released, and if the only remaining cycle is being deleted, the entire catalog entry is deleted.
- If the file is in use (open) within the requesting job, an abnormal status is returned.
- If other jobs are using the file at the time it is deleted, the file remains available to those jobs that are currently accessing the file until they detach the file.

- The job issuing the DELETE_FILE command can no longer access the file, that is, all outstanding attachments of the cycle to this job will be detached.
- Only a user with CONTROL permission can delete a file.
- For more information, see the NOS/VE System Usage manual.

Examples The following example deletes cycle number 88 of file DATA_FILE_1 that resides in the master catalog.

```
/delete_file $user.data_file_1.88 pw=pw_for_data_file_1
```

DELETE_FILE_CONNECTION Command

Purpose Deletes the connection between a subject file and a target file.

Format

DELETE_FILE_CONNECTION or
DELFC
STANDARD_FILE=file
FILE=file
STATUS=status variable

Parameters

STANDARD_FILE or **SF**

Specifies one of the following file names: $ECHO, $ERRORS, $INPUT, $LIST, $OUTPUT, $RESPONSE, or any other file which is connected. This parameter is required.

FILE or **F**

Specifies the name of the target file to be disconnected. This parameter is required.

Remarks

- The original connections for $RESPONSE cannot be deleted.
- The DISPLAY_FILE_CONNECTIONS command lists each subject file and its target files.
- For more information, see the NOS/VE System Usage manual.

Examples The following example deletes two subject file connections.

```
/delete_file_connection $echo echo_file
/delfc $response response_file
```

DELETE_FILE_PERMIT Command

Purpose Deletes an access control entry that was previously established for a file.

Format

DELETE_FILE_PERMIT or
DELFP
FILE=file
GROUP=keyword
FAMILY_NAME=name
USER=name
ACCOUNT=name
PROJECT=name
STATUS=status variable

Parameters

FILE or **F**

Specifies the file for which an access control entry is being deleted. This parameter is required.

GROUP or *G*

Specifies if the permit entry to be deleted applies to a specific user or group of users. The group selections are:

PUBLIC

All users regardless of family, account, project or user identifications.

FAMILY

All users in the specified family.

ACCOUNT

All users associated with the specified family and account identifications.

PROJECT

All users associated with the specified family, account, and project identifications.

USER

The user identified by the specified family and user identifications.

USER_ACCOUNT

The user identified by the specified family, account, and user identifications.

MEMBER

The user identified by the specified family, account, project, and user identifications.

Omission causes USER to be used.

FAMILY_NAME or *FN*

Specifies the family name associated with the permit entry being deleted. Omission causes the family name associated with the requesting job to be used if the GROUP selection indicates this parameter is applicable. If the GROUP selection indicates this parameter is not applicable, an abnormal status is returned.

USER or *U*

Specifies the user name associated with the permit entry being deleted. Omission causes the user name associated with the requesting job to be used if the GROUP selection indicates this parameter is applicable. If the GROUP selection indicates this parameter is not applicable, an abnormal status is returned.

ACCOUNT or *A*

Specifies the account name associated with the permit entry being deleted. Omission causes the account name associated with the requesting job to be used if the GROUP selection indicates this parameter is applicable. If the GROUP selection indicates this parameter is not applicable, an abnormal status is returned.

PROJECT or *P*

Specifies the project name associated with the permit entry being deleted. Omission causes the project name associated with the requesting job to be used if the GROUP selection indicates this parameter is applicable. If the GROUP selection indicates this parameter is not applicable an abnormal status is returned.

Remarks

- This request can be issued only by the owner of the file.
- For more information, see the NOS/VE System Usage manual.

Examples

The following example deletes a file permit for file DATA_FILE_1 in catalog CATALOG_1.

```
/delete_file_permit $user.catalog_1.data_file_0 ..
../group=user user=krj
/disc $user.catalog_1.data_file_0 permits
   PERMIT_GROUP: USER
     FAMILY: NVE, USER: DLH
     PERMITS: READ
     SHARE: READ
     APPLICATION_INFORMATION:
```

The permission previously established for user KRJ is removed, leaving only the permission for user DLH.

DELETE_REMOTE_VALIDATION Command

Purpose

Removes a remote validation established by a previous CREATE_REMOTE_VALIDATION command.

Format

DELETE_REMOTE_VALIDATION or
DELRV
LOCATION=list of name or **keyword**
STATUS=status variable

Parameters

LOCATION or **L**

Specifies the name of the remote location for which the validation is to be deleted. Values can be a list of remote location names or the keyword ALL. If ALL is specified, all lists of validation information established by the current job are deleted. If omitted, ALL is assumed.

Remarks

For more information, see the NOS/VE System Usage manual.

DELETE_VARIABLE Command

Purpose Deletes variable declarations from the current block.

Format

DELETE_VARIABLE or
DELETE_VARIABLES or
DELV
NAMES=list of name
STATUS=status variable

Parameters **NAMES** or **NAME** or **N**

Specifies the name(s) of the variable(s) whose declaration is to be removed. This parameter is required.

Remarks

- You can use the DISPLAY_VARIABLE_LIST command to list variables that are currently accessible.
- For more information, see the NOS/VE System Usage manual.

Examples The following example uses the DELETE_VARIABLE command to remove variables named COUNT and LOOPS.

```
/delete_variable (count,loops)
```

DESIGN_SCREEN Command

Purpose Enables you to enter the Screen Design Facility (SDF).

Format

DESIGN_SCREEN or
DESIGN_SCREENS or
DESS
LIBRARY=file
MODE=keyword
SCREEN_NAME=name
SOURCE=file
STATUS=status variable

Parameters *LIBRARY* or *L*

Specifies the name of the object library for the SDF session. All object screen modules for this SDF session are extracted from and/or placed in this library. The default library is SCREEN_LIBRARY in the current working catalog. The maximum number of screens you can store in

your library is 200. The object library used by SDF can be maintained by the CREATE_OBJECT_LIBRARY utility.

MODE or *M*

Indicates the session mode. Options are:

EDIT

Displays the Select screen.

LEARN

Guides you to the SDF online manual.

This parameter enables you to bypass the SDF Banner screen and go directly to the mode indicated by the keyword. If not specified, you select the mode from the SDF Banner screen.

SCREEN_NAME or *SN*

Specifies the name of an existing screen in your object library. This optional parameter causes immediate entry into an edit session at the named screen. You bypass the Select screen from which you select a screen. The MODE parameter takes precedence over the SCREEN_NAME parameter.

SOURCE or *S*

Specifies a local file name to write a record definition for any screen that is saved in the object library specified by the LIBRARY parameter. Record definitions are generated source code descriptions of all variables defined for the screen.

The definitions are written in the programming language of the application using the screen. Each definition is in the form of a source text record which can be used as input for SCU. The screen name is used as the name of the deck. If SOURCE is not specified, record definitions are written on the file $LOCAL.SOURCE.

Remarks

- For more information, see the Screen Design Facility manual.
- There are three ways to access SDF when you logon with the DESS command. First, you can enter the DESS command without parameters and use the SDF Banner screen. Second, you can enter the DESS command with the SCREEN_NAME parameter and immediately enter the EDIT mode with the specified screen displayed at your terminal. Third, you can enter the DESS command with the MODE parameter.

DETACH_FILE Command

Purpose Detaches one or more files from a job.

Format

DETACH_FILE or
DETACH_FILES or
DETF
FILES=list of file
STATUS=status variable

Parameters

FILES or **FILE** or **F**

Specifies the files to be detached. This parameter is required.

Remarks

- Following this command the files are no longer accessible to the job.
- The file must not be opened.
- For temporary files, this request also deletes the registration of the file in the temporary catalog.
- To return a file, all instances of open for the specified local file name must be closed. Standard files that reside in the $LOCAL catalog (such as $LIST) cannot be returned, because those files always have an outstanding instance of open within a job.
- If the file is a permanent mass storage file, the attachment of the file to the job ends with this request. The file's reserved mass storage space and its related file attributes and catalog information remain unaffected.

- If the file is a temporary mass storage file, its space is returned to the system and its attribute set is deleted.
- If you specify more than one file and an abnormal status is encountered when detaching one of the files, only those files which were specified previous to the file which returned an abnormal status are detached.
- For more information, see the NOS/VE System Usage manual.

Examples Following are examples of how to detach a single file (SCRATCH) and more than one file (TEST1,TEST2):

```
/detach_file file=scratch
/detach_files file=(test1,test2)
```

DETACH_JOB Command

Purpose Explicitly disconnects your terminal from the current job.

Format **DETACH_JOB** or
DETJ
STATUS=status variable

Remarks
- A disconnected terminal can be reconnected at any time with the ATTACH_JOB command.
- You can also disconnect your terminal by entering the network keystroke sequence of the network by which your terminal is connected to the system; the disconnection sequences are listed in this manual.
- For more information, see the NOS/VE System Usage manual.

Examples In the following NOS dual-state example from the NAM network, the interactive session is interrupted, disconnected, and then reconnected.

```
/detach_job

Welcome to the NOS/VE Software System.
Copyright Control Data 1983, 1987.
CYBER 855  Class SN2.  NOS/VE  14904 7P
March 20, 1987.  8:46 AM.

You have the following detached jobs:
$0855_0002_PDQ_0861

/display_job_status all
System_Supplied_Name     :   $0855_0002_pdq_0861
User_Supplied_Name       :   dlh_23
Originating_User         :   dlh
Originating_Family       :   nve
Job_Class                :   interactive
Job_Mode                 :   interactive command disconnect
Job_State                :   initiated
Operator_Action_Posted   :   no
Display_Message          :   fortran i=$user.source_file l=listing

System_Supplied_Name     :   $0855_0002_pdq_0862
User_Supplied_Name       :   dlh_23$
Originating_User         :   dlh
Originating_Family       :   nve
Job_Class                :   interactive
Job_Mode                 :   interactive connected
Job_State                :   initiated
Operator_Action_Posted   :   no
Display_Message          :   display_job_status all
/attach_job $0861
Job has been reconnected to this terminal
*Suspended - 1*
p/resume_command
/
```

DISPLAY_ACTIVE_TASKS Command

Purpose Displays the tasks which are executing within the current job.

Format

DISPLAY_ACTIVE_TASKS or
DISAT
OUTPUT=file
STATUS=status variable

Parameters *OUTPUT* or *O*

Specifies the file to which output is to be written. If the OUTPUT parameter is omitted, file $OUTPUT is used.

Remarks

- You can also execute this command at any time from an interactive terminal by entering:

```
(ncc) A
```

  Where (ncc) is your network command character.

- For more information, see the NOS/VE System Usage manual.

DISPLAY_APL_WORKSPACE Command

Remarks Reserved for site personnel, Control Data, or future use.

DISPLAY_BACKUP_LABEL_TYPE Command

Purpose Displays the current job default label type for a permanent file backup file which is assigned to a tape device.

Format

DISPLAY_BACKUP_LABEL_TYPE or
DISBLT
OUTPUT=file
STATUS=status variable

Parameters *OUTPUT* or *O*

Specifies the file into which information is written. Omission causes $OUTPUT to be used.

Remarks For more information, see the NOS/VE System Usage manual.

Examples The following example displays the current job default label type for a permanent file backup assigned to a tape device:

```
/display_backup_label_type
```

DISPLAY_CATALOG Command

Purpose Displays information about the files and catalogs registered in a specified catalog or about the access control list established at a catalog level.

Format

DISPLAY_CATALOG or
DISC
CATALOG=file
DISPLAY_OPTIONS=keyword
OUTPUT=file
DEPTH=integer or *keyword*
STATUS=status variable

Parameters *CATALOG* or *C*

Specifies the catalog from which information is to be displayed. Omission causes the current working catalog to be used.

DISPLAY_OPTIONS or *DISPLAY_OPTION* or *DO*

Specifies the type of display being requested. Options are:

IDENTIFIER (ID, I)

Selects a display of the file or catalog name and type (file or catalog) of each entry in the specified catalog.

FILE (F)

Selects a display of a summary description of files registered in the specified catalog.

PERMIT (PERMITS, P)

Selects a display of access control entries that describe the access permissions established at this catalog level.

CONTENT (CONTENTS, C)

Displays the amount of disk space in bytes that is occupied by the files within the catalog, the size of each file, and the damage condition of each cycle (if damaged). The information to be displayed is specified with the DEPTH parameter.

Omission causes IDENTIFIER to be used.

OUTPUT or *O*

Identifies the file to which information is displayed and, optionally, specifies how the file is to be positioned prior to use. This parameter cannot specify a file associated with a remote family. Omission causes $OUTPUT to be used.

DEPTH or *D*

Specifies the amount of information that is to be displayed when the CONTENT display option is selected. If a depth of 1 is specified, a one-line summary of the catalog is displayed; if a depth of 2 is specified, a breakdown of the files and subcatalogs contained in the catalog is displayed; and if a depth of 3 or more is specified, a further breakdown of the catalogs and files contained within the subcatalogs of the catalog is displayed. Specifying the keyword ALL displays information about all subcatalogs and files contained in a catalog. If the DEPTH parameter is omitted, a depth of 2 is assumed.

Remarks

- If the catalog belongs to another user, the display includes only information for files or catalogs to which the requesting user is permitted access (that is, permitted for any of the access modes).
- For more information, see the NOS/VE System Usage manual.

Examples

The following example displays information for the master catalog using the ID option.

```
/display_catalog $user do=id
 CATALOG: CATALOG_1
    FILE: DATA_FILE_1
    FILE: EPILOG
    FILE: PROLOG
```

The following example displays information for the master catalog using the FILE option.

```
/disc $user do=file
DATA_FILE_1
    NUMBER OF CYCLES:  1, ACCOUNT: D5927, PROJECT: P693N354
EPILOG
    NUMBER OF CYCLES:  1, ACCOUNT: D5923, PROJECT: P693N354
PROLOG
    NUMBER OF CYCLES:  1, ACCOUNT: D5923, PROJECT: P693N354
```

The following example displays information for the master catalog using the PERMITS option.

```
/disc $user do=permits
   PERMIT_GROUP: USER
     FAMILY: NVE, USER: SDH
     PERMITS: READ, SHORTEN, APPEND, MODIFY, EXECUTE,
              CYCLE, CONTROL
     SHARE: NONE
     APPLICATION_INFORMATION:
```

The following example shows the catalog information displayed after a damage condition was encountered on file DATA_FILE_1. Along with the file's cycle number (88), the system indicates only that the file contains greater than zero bytes.

```
/display_catalog $user do=content
sdh                    3,978 bytes in 3 files
  data_file_1              0 bytes in cycle 88
  prolog                 991 bytes
  scu_editor_prolog    2,987 bytes
```

DISPLAY_CATALOG_ENTRY Command

Purpose Displays catalog information about a file, its usage, or its access control list.

Format

DISPLAY_CATALOG_ENTRY or
DISCE
FILE=file
DISPLAY_OPTIONS=keyword
OUTPUT=file
DEPTH=integer or *keyword*
STATUS=status variable

Parameters

FILE or **F**

Specifies the file for which catalog information is to be displayed. Omission of the cycle causes a display for each cycle of the file. This parameter is required.

DISPLAY_OPTIONS or *DISPLAY_OPTION* or *DO*

Specifies the type of display being requested. Options are:

LOG (L)

Selects a display of file usage information contained in the usage log. If the file is not owned by the requesting user, only the log entry for the requestor is displayed.

PERMIT (PERMITS, P)

Selects a display of access control entries that describe the access permission to a file. The access control entries are created using the CREATE_FILE_PERMIT command.

DESCRIPTOR (D)

Selects a display of the catalog information that describes the file and the file cycles.

CYCLE (CYCLES, C)

Displays the amount of disk space in bytes that is occupied by a file cycle as well as the damage condition of the cycle (if it is damaged). Whether a summary for all cycles or information for each cycle is to be displayed is specified with the DEPTH parameter.

Omission causes DESCRIPTOR to be used.

OUTPUT or *O*

Identifies the file to which information is displayed and, optionally, specifies how the file is to be positioned prior to use. This parameter cannot specify a file associated with a remote family. Omission causes $OUTPUT to be used.

DEPTH or *D*

Specifies the amount of information that is to be displayed when the CYCLE display option is selected. If a depth of 1 is specified, a one-line summary of the file is displayed; if a depth of 2 is specified, a breakdown of each cycle of the file is displayed. Specifying the keyword ALL, displays information about all cycles contained in a file. If the DEPTH parameter is omitted, a depth of 2 is assumed.

Remarks

- If the file belongs to another user, the display is provided only if the requesting user is permitted access for any of the access modes.
- For more information, see the NOS/VE System Usage manual.

Examples The following example displays information for file DATA_FILE_1 in the master catalog using the LOG display option.

```
/display_catalog_entry $user.data_file_1 do=log
     DATE AND TIME: 1986-06-06 10:32:36.072,
     FAMILY: NVE, USER: SDH,
     ACCESS COUNT: 3, LAST CYCLE: 88
```

The following example displays information for file DATA_FILE_1 in the master catalog using the PERMITS display option.

```
/disce $user.data_file_1 do=permits
    NO PERMITS
```

The following example displays information for file DATA_FILE_1 in the master catalog using the DESCRIPTOR display option.

```
/disce $user.data_file_1 do=descriptor
    NUMBER OF CYCLES:  1, ACCOUNT: D5927, PROJECT: P693N354
    PASSWORD: PW_FOR_DATA_FILE_1, LOG SELECTION: TRUE
    CYCLE NUMBER:      1, ACCESS COUNT: 1,
    CREATION DATE AND TIME: 1986-06-06 10:32:18.550,
    LAST ACCESS DATE AND TIME: 1986-06-06 10:32:18.647,
    LAST MODIFICATION DATE AND TIME: 1986-06-06 10:32:18.647,
    EXPIRATION DATE: 1986-12-11
```

The following example displays information for cycle 88 of file DATA_FILE_1 after a damage condition has occurred:

```
/display_catalog_entry $user.data_file_1 do=cycles
data_file_1    0 bytes in cycle 88
--  cycle 88              -- respf modification mismatch
```

DISPLAY_COMMAND_INFORMATION Command

Purpose Displays information about a command and its parameters in the defined order of that command's parameters. The information includes the names and abbreviations of parameters, their types (including allowed keywords), and the default values (or an indication that the parameter is required).

Format

DISPLAY_COMMAND_INFORMATION or
DISCP or
DISPLAY_COMMAND_PARAMETER or
DISPLAY_COMMAND_PARAMETERS or
DISCI
COMMAND=command
OUTPUT=file
STATUS=status variable

Parameters

COMMAND or **C**

Specifies the name of the command for which information is being sought. This parameter is required.

OUTPUT or *O*

Specifies the file to which information is written. Omission casues $OUTPUT to be used.

Remarks

- The DISPLAY_COMMAND_INFORMATION command returns information for all system-supplied commands, SCL procedures, user programs, and utility subcommands. You can use it for any command that can be called at the point where the DISPLAY_COMMAND_INFORMATION command is issued. For instance, to get information about a Source Code Utility (SCU) subcommand, you must be in SCU when you issue the DISPLAY_COMMAND_INFORMATION command.

- This command is intended as a memory aid to help you recall both the required entries for a commmand and the optional parameters. The online manual or printed manual should be consulted for complete information about a command and its parameters.

- For more information, see the NOS/VE System Usage manual.

Examples The following example shows command information for the CREATE_FILE command.

```
/display_command_information command=create_file
file, f                : file = $required
local_file_name,lfn    : name = $optional
password, pw           : name = none
retention, r           : integer 1 .. 999 = 999
log, l                 : boolean = false
status                 : var of status = $optional
```

The FILE parameter accepts a file name as a value; there is no default value because the parameter is required. The RETENTION parameter is an integer in the range 1 through 99, the parameter is optional and the default value for the parameter is 999.

DISPLAY_COMMAND_LIST Command

Purpose Displays information about the command list.

Format **DISPLAY_COMMAND_LIST** or **DISCL**

DISPLAY_OPTIONS=list of keyword
OUTPUT=file
STATUS=status variable

Parameters *DISPLAY_OPTIONS* or *DISPLAY_OPTION* or *DO*

Specifies a display option, entries can be:

ENTRY (E)

Displays the names of all entries in the command list.

SEARCH_MODE (SM)

Displays the command list search mode.

ALL

Displays the names of all entries in the command list and the command list search mode.

Omission causes ENTRY to be used.

OUTPUT or *O*

Identifies the file to which the information is written and, optionally, specifies how the file is to be positioned prior to use. Omission causes $OUTPUT to be used.

Remarks

- Control Data is not responsible for the proper functioning of statements that are displayed by the DISPLAY_COMMAND_LIST command but are not documented in the NOS/VE manuals.
- For more information, see the NOS/VE System Usage manual.

Examples 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
```

DISPLAY_COMMAND_LIST_ENTRY Command

Purpose Displays information about one or more entries in a command list.

Format

DISPLAY_COMMAND_LIST_ENTRY or **DISCLE**
ENTRY=list of file or *keyword*
DISPLAY_OPTIONS=list of keyword
OUTPUT=file
STATUS=status variable

Parameters

ENTRY or *ENTRIES* or *E*

Selects the command list entry or entries you want to display. Any entry that can be put in the command list can be selected even if the entry is not presently in the command list. In addition to entering the name of a particular command list entry, you can also specify one of the following options to select a display based on entry type.

CONTROL_STATEMENT (CS)

Displays the SCL control statements and commands.

FIRST (F)

Displays the first entry in the command list, allowing you to select a display of the subcomands supplied by an active utility.

ALL

Displays the entire command list including the control statements.

Omission causes FIRST to be used.

DISPLAY_OPTIONS or *DISPLAY_OPTION* or *DO*

Selects the content of the output. You may specify one of the following options:

NAME (N)

Displays the names of commands, functions, or control statements in the command list. The names displayed depend on the other display options you have chosen.

ALL_NAMES (AN)

Displays all names for commands, functions, and control statements in the command list. These names include abbreviations and aliases.

COMMAND (C)

Displays information about individual commands within the command list entries. Commands that reside within catalogs are not displayed.

FUNCTION (F)

Displays information about individual functions within the command list entries.

ALL

Selects all the other options.

Omission of the DISPLAY_OPTION parameter causes COMMAND and NAME to be used. If you do not specify either COMMAND or FUNCTION as one of the DISPLAY_OPTION choices, COMMAND is automatically selected.

OUTPUT or *O*

Identifies the file to which information is written and, optionally, specifies how the file is to be positioned prior to use. Omission causes $OUTPUT to be used.

Remarks For more information, see the NOS/VE System Usage manual.

DISPLAY_CONNECTION_ATTRIBUTES Command

Purpose Displays either all or a specified set of the terminal file's connection attributes.

Format

DISPLAY_CONNECTION_ATTRIBUTES or
DISPLAY_CONNECTION_ATTRIBUTE or
DISPLAY_TERM_CONN_ATTRIBUTE or
DISPLAY_TERM_CONN_ATTRIBUTES or
DISTCA or
DISCA
TERMINAL_FILE_NAME=file
DISPLAY_OPTIONS=list of name
OUTPUT=file
STATUS=status variable

Parameters **TERMINAL_FILE_NAME** or **TFN**

Specifies the terminal file name. This parameter is required.

DISPLAY_OPTIONS or *DISPLAY_OPTION* or *DO*

Specifies the names of the attributes you want displayed.

See the CHANGE_CONNECTION_ATTRIBUTES (CHACA) command description for descriptions of the attributes.

Omission or the keyword ALL causes all attributes to be displayed.

OUTPUT or *O*

Specifies the name of the file on which the information is to be displayed and, optionally, how the file is to be positioned prior to use. Omission causes $OUTPUT to be used.

Remarks For more information, see the NOS/VE System Usage manual.

DISPLAY_FILE Command

Purpose Displays a file in hexadecimal and/or ASCII form.

Format

DISPLAY_FILE or
DISF
INPUT=file
OUTPUT=file
FORMATS=list of keyword
BYTE_ADDRESSES=list of range of integer
STATUS=status variable

Parameters

INPUT or **I**

Specifies the file from which data is to be obtained and, optionally, specifies how the file is to be positioned prior to use. The file must not be open or attached with a SHARE_MODE or OPEN_SHARE_MODE parameter value of APPEND, MODIFY, or SHORTEN.

Data is displayed from the open position or byte address until end-of-information is reached. This parameter is required.

OUTPUT or *O*

Specifies the file to which data is to be written and, optionally, specifies how the file is to be positioned prior to use. Omission causes $OUTPUT to be used.

FORMATS or *FORMAT* or *F*

Specifies whether to display in ASCII, hexadecimal, or both. Options are:

ASCII

Display in ASCII (appears in 2-byte quantities).

HEX

Display in hexadecimal (appears in 1-byte quantities).

Omission of the parameter or selection of both the ASCII and HEX values causes both the hexadecimal and its printable ASCII equivalent to be displayed.

BYTE_ADDRESSES or *BYTE_ADDRESS* or *BA*

Specifies the range of byte addresses to be displayed from the input file. This parameter can be used only if the input file is assigned to a disk device. Omission causes the file to be displayed starting at the open position.

Remarks For more information, see the NOS/VE System Usage manual.

Examples The following example displays the internal contents of the entire file FILE_1. Only the ASCII format is requested.

```
/display_file 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.

```
/disf file_1 f=(ascii,hex)
BYTE ADDRESS          HEXADECIMAL                ASCII
         0 0000000000001800 0G0000000001e5468             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 next 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.

```
/disf 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
```

DISPLAY_FILE_ATTRIBUTES Command

Purpose Specifies the files for which the selected attributes are to be displayed.

Format

DISPLAY_FILE_ATTRIBUTES or
DISPLAY_FILE_ATTRIBUTE or
DISFA
FILE=list of file
DISPLAY_OPTIONS=list of name
OUTPUT=file
STATUS=status variable

Parameters

FILE or **F**

Specifies the files for which the selected attributes are to be displayed. This parameter is required.

DISPLAY_OPTIONS or *DISPLAY_OPTION* or *DO*

Specifies the attribute that should be displayed. If ALL is specified, all file attributes are displayed. If the keyword value SOURCE is specified, a description of how the attribute value was defined is provided.

The following is a list of attribute names (and their abbreviations) whose values may be displayed with the DISPLAY_FILE_ATTRIBUTE command:

ACCESS_MODE (AM)
AVERAGE_RECORD_LENGTH (ARL)
BLOCK_TYPE (BT)
CHARACTER_CONVERSION (CC)
COLLATE_TABLE_NAME (CTN)
COMPRESSION_PROCEDURE_NAME (CPN)
DATA_PADDING (DP)
EMBEDDED_KEY (EK) or
ERROR_EXIT_NAME (EEN)
ERROR_EXIT_PROCEDURE_NAME (EEPN)
ERROR_LIMIT (EL)
ESTIMATED_RECORD_COUNT (ERC)
FILE_ACCESS_PROCEDURE (FAP) or
FILE_ACCESS_PROCEDURE_NAME (FAPN)
FILE_CONTENTS (FC)
FILE_LIMIT (FL)
FILE_ORGANIZATION (FO)
FILE_PROCESSOR (FP)

FILE_STRUCTURE (FS)
FORCED_WRITE (FW)
HASHING_PROCEDURE_NAME (HPN)
INDEX_LEVEL (IL)
INDEX_PADDING (IP)
INITIAL_HOME_BLOCK_COUNT (IHBC)
INTERNAL_CODE (IC)
KEY_LENGTH (KL)
KEY_POSITION (KP)
KEY_TYPE (KT)
LINE_NUMBER (LN)
LOCK_EXPIRATION_TIME (LET)
MAXIMUM_BLOCK_LENGTH (MAXBL)
MAXIMUM_RECORD_LENGTH (MAXRL)
MESSAGE_CONTROL (MC)
MINIMUM_BLOCK_LENGTH (MINBL)
MINIMUM_RECORD_LENGTH (MINRL)
OPEN_POSITION (OP)
PADDING_CHARACTER (PC)
PAGE_FORMAT (PF)
PAGE_LENGTH (PL)
PAGE_WIDTH (PW)
PRESET_VALUE (PV)
RECORD_LIMIT (RL)
RECORD_TYPE (RT)
RECORDS_PER_BLOCK (RPB)
STATEMENT_IDENTIFIER (SI)
USER_INFORMATION (UI)

The following attribute names may be used to query information about the file which was not provided by the SET_FILE_ATTRIBUTE command:

APPLICATION_INFORMATION (AI)

Specifies information used by application programs for additional access controls they impose.

SIZE (S)

Gives the file length in bytes.

GLOBAL_ACCESS_MODE (GAM)

Gives the modes of access the job is permitted to the file. The possible modes of access are READ, WRITE, APPEND, MODIFY, SHORTEN, and EXECUTE.

GLOBAL_FILE_ADDRESS (GFA)

Indicates the current byte address attained by the last file access request issued against the file.

GLOBAL_FILE_NAME (GFN)

Specifies the unique file name identifying the file. The system generates this name when it creates the file.

GLOBAL_FILE_POSITION (GFP)

Indicates the file position attained by the last file access request issued against the file. The possible file positions:

BOI

Beginning-of-information.

BOP

Beginning-of-partition.

MID_RECORD

Positioned between the beginning and end of a record.

EOR

End-of-record.

EOP

End-of-partition.

EOI

End-of-information.

GLOBAL_SHARE_MODE (GSM)

Indicates whether the file can be shared among other jobs and which modes of access sharing are possible. Possible modes are NONE (implies the job has exclusive use of the file), READ, WRITE, APPEND, MODIFY, SHORTEN, and EXECUTE.

PERMANENT (P)

Indicates (via a boolean value) whether or not the file is permanent.

RING_ATTRIBUTES (RA)

Indicates the ring attributes that are preserved with the file.

Omission causes these options to be used:

FILE_CONTENTS
FILE_PROCESSOR
FILE_STRUCTURE
GLOBAL_ACCESS_MODE
PERMANENT
SIZE

OUTPUT or *O*

Specifies the file upon which the information is to be displayed and, optionally, specifies how the file is to be positioned prior to use. This parameter cannot specify a file associated with a remote family. Omission causes $OUTPUT to be used.

Remarks

- Each display entry includes the file name and the list of requested attribute names and their associated values.
- For the DISPLAY_OPTION parameter, the SOURCE keyword must be used with another keyword or an abnormal status is returned.
- The SCL function $FILE can also be used to obtain certain file attributes.
- For more information, see the NOS/VE System Usage manual.

Examples 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)
Application_Information      : none
Average_Record_Length        : 0
Block_Type                   : system_specified
Character_Conversion         : no
Collate_Table_Name           : none
Compression_Procedure_Name   : none
Data_Padding                 : 0
Embedded_Key                 : yes
Error_Exit_Name              : none
Error_Limit                  : 0
Estimated_Record_Count       : 0
File_Access_Procedure        : none
File_Contents                : unknown
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             : $0000000p2s0000d19800812t000000
Global_File_Position         : boi
Global_Share_Mode            : none
Hashing_Procedure_Name       :  /(amp$system_hashing_procedure)
Index_Levels                 : 2
Index_Padding                : 0
Initial_Home_Block_Count     :  /1
Internal_Code                : ascii
Key_Length                   : 0
Key_Position                 : 0
Key_Type                     : uncollated
Line_Number                  :  /("Length" 1, "Location" 1)
Loading_Factor               : 90
Lock_Expiration_Time         : 60000
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
```

DISPLAY_FILE_CONNECTIONS Command

Purpose Displays the names of the files currently connected to the subject files.

Format

DISPLAY_FILE_CONNECTIONS or
DISPLAY_FILE_CONNECTION or
DISFC
STANDARD_FILES=list of file or *keyword*
OUTPUT=file
STATUS=status variable

Parameters

STANDARD_FILES or *STANDARD_FILE* or *SF*

Specifies any or all of the following subject file names: $ECHO, $ERRORS, $INPUT, $LIST, $OUTPUT, $RESPONSE, or any other file which is connected. If the keyword ALL is specified or if this parameter is omitted, the file connections for all subject files are listed.

OUTPUT or *O*

Identifies the file on which to write the file names associated with the job's current subject file connections and optionally, specifies how the file is to be positioned prior to use. Omission causes $OUTPUT to be used.

Remarks For more information, see the NOS/VE System Usage manual.

Examples The following example displays the file connections for subject files $RESPONSE, $ECHO, and $OUTPUT.

```
/display_file_connection ($response,$echo,$output)
:$LOCAL.$RESPONSE.1 is connected to: :$LOCAL.$JOB_LOG.1, :$LOCAL.OUTPUT.1.
:$LOCAL.$ECHO.1 is not connected to any files.
:$LOCAL.$OUTPUT.1 is connected to: :$LOCAL.OUTPUT.1.
```

The following example displays the file connections for all of the subject files.

```
/disfc all
:$LOCAL.$ECHO.1 is not connected to any files.
:$LOCAL.$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.$JOB_LOG.1, :$LOCAL.OUTPUT.1.
```

DISPLAY_FUNCTION_INFORMATION Command

Purpose Displays information about a function.

Format

DISPLAY_FUNCTION_INFORMATION or
DISFI
FUNCTION=function
OUTPUT=file
STATUS=status variable

Parameters

FUNCTION or **F**

Specifies the function. This parameter is required.

OUTPUT or *O*

Specifies the file to which the function information is written. The default file is OUTPUT.

Remarks For more information, see the NOS/VE System Usage manual.

Examples The following example requests information about the $DATE function:

```
/display_function_information function=$date
parameter 1 : key month, mdy, dmy, iso, isod, ..
                 ordinal, default = default
```

This description shows the following:

- The $DATE function has one parameter.
- Possible entries for this parameter are one of the following keywords:

  MONTH
  MDY
  DMY
  ISO
  ISOD
  ORDINAL
  DEFAULT

- The parameter is optional. If you omit this parameter, the system assumes DEFAULT is in effect.

DISPLAY_JOB_ATTRIBUTE Command

Purpose Displays the attributes of your current job.

Format

DISPLAY_JOB_ATTRIBUTE or
DISPLAY_JOB_ATTRIBUTES or
DISJA
DISPLAY_OPTION=list of keyword
OUTPUT=file
STATUS=status variable

Parameters *DISPLAY_OPTION* or *DISPLAY_OPTIONS* or *DO*

Specifies the information you want to display. The keywords are:

ALL

Displays all information. This is the default.

COMMENT_BANNER (CB)

Displays the COMMENT_BANNER job attribute.

CONTROL_FAMILY (CF)

Displays the family name of the control user.

CONTROL_USER (CU)

Displays the user name of the control user.

COPIES (C)

Displays the COPIES job attribute.

CYCLIC_AGING_INTERVAL (CAI)

Displays the CYCLIC_AGING_INTERVAL job attribute.

DETACHED_JOB_WAIT_TIME (DJWT)

Displays the DETACHED_JOB_WAIT_TIME job attribute.

DEVICE (D)

Displays the DEVICE job attribute.

DISPATCHING_PRIORITY (DP)

Displays the DISPATCHING_PRIORITY job attribute.

EARLIEST_PRINT_TIME (EPT)

Displays the EARLIEST_PRINT_TIME job attribute.

EARLIEST_RUN_TIME (ERT)

Displays the EARLIEST_RUN_TIME job attribute.

EXTERNAL_CHARACTERISTICS (EC)

Displays the EXTERNAL_CHARACTERISTICS job attribute.

FORMS_CODE (FC)

Displays the FORMS_CODE job attribute.

JOB_ABORT_DISPOSITION (JAD)

Displays the JOB_ABORT_DISPOSITION job attribute.

JOB_CLASS (JC)

Displays the job class of your job.

JOB_MODE (JM)

Displays your job's job mode. Returnable values are INTERACTIVE or BATCH.

JOB_QUALIFIER (JOB_QUALIFIERS or JQ)

Displays from zero to five site-defined names that may be used to limit your 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 support this capability.

JOB_RECOVERY_DISPOSITION (JRD)

Displays the JOB_RECOVERY_DISPOSITION job attribute.

JOB_SIZE (JS)

Displays the size (in bytes) of your job's input file.

JOB_SUBMISSION_TIME (QUEUED_TIME or JST or QT)

Displays the time when your job arrived in the input queue.

LATEST_PRINT_TIME (LPT)

Displays the latest time that output files created by your job are to be printed. A value of NONE indicates there are no print restrictions.

LATEST_RUN_TIME (LRT)

Displays the latest time that your job may be initiated. If your job is not initiated by the specified time, your job is discarded. If no value is displayed, there is no restriction. For this release, this value is always NONE.

LOGIN_ACCOUNT (LA)

Displays the account name under which your job is scheduled and run.

LOGIN_FAMILY (LF)

Displays the family name under which your job is scheduled and run.

LOGIN_PROJECT (LP)

Displays the project name under which your job is scheduled and run.

LOGIN_USER (LU)

Displays the user name under which your job is scheduled and run.

MAXIMUM_WORKING_SET (MAXWS)

Displays the MAXIMUM_WORKING_SET job attribute.

MINIMUM_WORKING_SET (MINWS)

Displays the MINIMUM_WORKING_SET job attribute.

OPERATOR_FAMILY (DESTINATION_FAMILY or OF or DF)

Displays the OPERATOR_FAMILY job attribute.

OPERATOR_USER (STATION_OPERATOR or OU or SO)

Displays the OPERATOR_USER job attribute.

ORIGINATING_APPLICATION_NAME (OAN)

Displays the name of the application from which your job was entered in the system.

OUTPUT_CLASS (OC)

Displays the OUTPUT_CLASS job attribute.

OUTPUT_DESTINATION (ODE)

Displays the OUTPUT_DESTINATION job attribute.

OUTPUT_DESTINATION_USAGE (ODU or DU)

Displays the OUTPUT_DESTINATION_USAGE job attribute.

OUTPUT_DISPOSITION (ODI)

Displays the OUTPUT_DISPOSITION job attribute.

OUTPUT_PRIORITY (OP)

Displays the OUTPUT_PRIORITY job attributes.

PAGE_AGING_INTERVAL (PAI)

Displays the PAGE_AGING_INTERVAL job attribute.

PURGE_DELAY (PD)

Reserved for future use.

REMOTE_HOST_DIRECTIVE (RHD)

Displays the REMOTE_HOST_DIRECTIVE job attribute.

ROUTING_BANNER (RB)

Displays the ROUTING_BANNER job attribute.

SENSE_SWITCHES (SS)

Displays the settings of the job's sense switches.

SERVICE_CLASS (SC)

Displays the name of the job's service class.

SITE_INFORMATION (SI)

Displays the SITE_INFORMATION string associated with all output files created by your job.

STATION (S)

Displays the STATION job attribute.

SYSTEM_JOB_NAME (SJN)

Displays the name assigned to your job by the system.

USER_INFORMATION (UI)

Displays the USER_INFORMATION job attribute.

USER_JOB_NAME (UJN)

Displays the user-supplied name of your job.

VERTICAL_PRINT_DENSITY (VPD)

Displays the VERTICAL_PRINT_DENSITY job attribute.

VFU_LOAD_PROCEDURE (VLP)

Displays the VFU_LOAD_PROCEDURE job attribute.

OUTPUT or *O*

Specifies the file to which the requested information will be written. You can specify a file position as part of the name. If omitted, $OUTPUT is assumed.

Remarks

- If you enter this command without parameters, the system displays all of your job's attributes.
- The SCL function $JOB can be used to determine certain attributes of your job.
- See the CHANGE_JOB_ATTRIBUTE command for information on how to change the values of the attributes displayed with this command.
- For more information, see the NOS/VE System Usage manual.

Examples The following example shows the job attributes for user job name SARETT.

```
/display_job_attributes do=all
Comment_Banner                  : ' '
Control_Family                  : nve
Control_User                    : sarett
Copies                          : 1
Cyclic_Aging_Interval           : 1000000000
Detached_Job_Wait_Time          : 3600
Device                          : automatic
Dispatching_Priority            : p5
Earliest_Print_Time             : none
Earliest_Run_Time               : none
External_Characteristics        : 'NORMAL'
Forms_Code                      : 'NORMAL'
Job_Abort_Disposition           : terminate
Job_Class                       : interactive
Job_Mode                        : interactive connected
Job_Qualifier                   : []
Job_Recovery_Disposition        : continue
Job_Size                        : 0
Job_Submission_Time             : 1987-07-31.14:51:41
Latest_Print_Time               : none
Latest_Run_Time                 : none
Login_Account                   : d1257
Login_Family                    : nve
Login_Project                   : p83a2821
Login_User                      : sarett
Maximum_Working_Set             : 1000
Minimum_Working_Set             : 20
Operator_Family                 : nve
Operator_User                   : sarett
Originating_Application_Name    : osa$dual_state_interactive
Output_Class                    : normal
Output_Destination              : 'NVE'
Output_Destination_Usage        : dual_state
Output_Disposition              : printer
Output_Priority                 : low
Page_Aging_Interval             : 50000
Purge_Delay                     : none
Remote_Host_Directive           : ' '
Routing_Banner                  : ' '
Sense_Switches                  : []
Service_Class                   : interactive
Site_Information                : ' '
Station                         : automatic
System_Job_Name                 : $0990_0102_aad_1367
User_Information                : ' '
User_Job_Name                   : sarett
Vertical_Print_Density          : file
VFU_Load_Procedure              : none
```

SITE_INFORMATION (SI)

Displays the SITE_INFORMATION string associated with all output files created by your job.

STATION (S)

Displays the STATION job attribute.

SYSTEM_JOB_NAME (SJN)

Displays the name assigned to your job by the system.

USER_INFORMATION (UI)

Displays the USER_INFORMATION job attribute.

USER_JOB_NAME (UJN)

Displays the user-supplied name of your job.

VERTICAL_PRINT_DENSITY (VPD)

Displays the VERTICAL_PRINT_DENSITY job attribute.

VFU_LOAD_PROCEDURE (VLP)

Displays the VFU_LOAD_PROCEDURE job attribute.

OUTPUT or *O*

Specifies the file to which the requested information will be written. You can specify a file position as part of the name. If omitted, $OUTPUT is assumed.

Remarks

- If you enter this command without parameters, the system displays all of your job's attributes.
- The SCL function $JOB can be used to determine certain attributes of your job.
- See the CHANGE_JOB_ATTRIBUTE command for information on how to change the values of the attributes displayed with this command.
- For more information, see the NOS/VE System Usage manual.

Examples The following example shows the job attributes for user job name SARETT.

```
/display_job_attributes do=all
Comment_Banner                  : ' '
Control_Family                  : nve
Control_User                    : sarett
Copies                          : 1
Cyclic_Aging_Interval           : 1000000000
Detached_Job_Wait_Time          : 3600
Device                          : automatic
Dispatching_Priority            : p5
Earliest_Print_Time             : none
Earliest_Run_Time               : none
External_Characteristics        : 'NORMAL'
Forms_Code                      : 'NORMAL'
Job_Abort_Disposition           : terminate
Job_Class                       : interactive
Job_Mode                        : interactive connected
Job_Qualifier                   : []
Job_Recovery_Disposition        : continue
Job_Size                        : 0
Job_Submission_Time             : 1987-07-31.14:51:41
Latest_Print_Time               : none
Latest_Run_Time                 : none
Login_Account                   : d1257
Login_Family                    : nve
Login_Project                   : p83a2821
Login_User                      : sarett
Maximum_Working_Set             : 1000
Minimum_Working_Set             : 20
Operator_Family                 : nve
Operator_User                   : sarett
Originating_Application_Name    : osa$dual_state_interactive
Output_Class                    : normal
Output_Destination              : 'NVE'
Output_Destination_Usage        : dual_state
Output_Disposition              : printer
Output_Priority                 : low
Page_Aging_Interval             : 50000
Purge_Delay                     : none
Remote_Host_Directive           : ' '
Routing_Banner                  : ' '
Sense_Switches                  : []
Service_Class                   : interactive
Site_Information                : ' '
Station                         : automatic
System_Job_Name                 : $0990_0102_aad_1367
User_Information                : ' '
User_Job_Name                   : sarett
Vertical_Print_Density          : file
VFU_Load_Procedure              : none
```

DISPLAY_JOB_ATTRIBUTE_DEFAULT Command

Purpose Displays the current settings of the system default values for job attributes.

Format **DISPLAY_JOB_ATTRIBUTE_DEFAULT** or
DISPLAY_JOB_ATTRIBUTE_DEFAULTS or
DISJAD

JOB_MODE=list of keyword
DISPLAY_OPTION=list of keyword
OUTPUT=file
STATUS=status variable

Parameters *JOB_MODE* or *JOB_MODES* or *JM*

Specifies the job modes for which the system default job attribute values are to be displayed. Options are:

ALL

Display the system default job attribute values for both interactive and batch jobs. This is the default.

BATCH

Display the system default job attribute values for batch jobs.

INTERACTIVE

Display the system default job attribute values for interactive jobs.

DISPLAY_OPTION or *DISPLAY_OPTIONS* or *DO*

Specifies what system default job attributes are to be displayed. Keywords are:

ALL

Displays all of the following keyword information. This is the default DISPLAY_OPTION value.

CPU_TIME_LIMIT (CTL)

Displays the system default CPU_TIME_LIMIT job attribute.

JOB_ABORT_DISPOSITION (JAD)

Displays the system default JOB_ABORT_ DISPOSITION job attribute.

JOB_CLASS (JC)

Displays the system default job class attribute for your job.

JOB_QUALIFIER (JOB_QUALIFIERS, JQ)

Displays the system default JOB_QUALITIER job attribute.

JOB_RECOVERY_DISPOSITION (JRD)

Displays the system default JOB_RECOVERY_ DISPOSITION job attribute.

LOGIN_FAMILY (LF)

Displays the system default LOGIN_FAMILY job attribute.

MAGNETIC_TAPE_LIMIT (MTL)

Displays the system default MAGNETIC_TAPE_ LIMIT job attribute.

MAXIMUM_WORKING_SET (MAXWS)

Displays the system default MAXIMUM_WORKING_ SET job attribute.

OUTPUT_CLASS (OC)

Displays the system default OUTPUT_CLASS job attribute.

OUTPUT_DESTINATION_USAGE (ODU)

Displays the system default OUTPUT_ DESTINATION_USAGE job attribute.

SITE_INFORMATION (SI)

Displays the system default SITE_INFORMATION job attribute.

SRU_LIMIT (SL)

Displays the system default SRU_LIMIT job attribute.

STATION (S)

Displays the system default STATION job attribute.

VERTICAL_PRINT_DENSITY (VPD)

Displays the system default VERTICAL_PRINT_ DENSITY job attribute.

OUTPUT or *O*

Specifies the file to which information is written. Omission causes $OUTPUT to be used.

Remarks

- This command displays only those job attributes having system default values that may be defined by your site. Note that some job attributes do not have system default values.
- Some of the values displayed by this command may be changed using the CHANGE_JOB_ATTRIBUTE command.
- For more information, see the NOS/VE System Usage manual.

Examples

The following example shows the default job attributes for your system.

```
/display_job_attribute_default display_options=all
 Job_Mode: BATCH
 CPU_Time_Limit              : unlimited
 Job_Abort_Disposition       : terminate
 Job_Class                   : batch
 Job_Qualifier               : []
 Job_Recovery_Disposition    : continue
 Login_Family                : nve
 Magnetic_Tape_Limit         : unspecified
 Maximum_Working_Set         : 1000
 Output_Class                : normal
 Output_Destination_Usage    : dual_state
 Site_Information            : 'CYBER 995 Class 102'
 SRU_limit                   : unlimited
 Station                     : ve_printer_109
 Vertical_Print_Density      : file

 Job_Mode: INTERACTIVE
 CPU_Time_Limit              : unlimited
 Job_Abort_Disposition       : terminate
```

```
Job_Class                    : interactive
Job_Qualifier                : []
Job_Recovery_Disposition     : continue
Login_Family                 : nve
Magnetic_Tape_Limit          : unspecified
Maximum_Working_Set          : 1000
Output_Class                 : normal
Output_Destination_Usage     : dual_state
Site_Information             : 'CYBER 995 Class 102'
SRU_limit                    : unlimited
Station                      : ve_printer_109
Vertical_Print_Density       : file
```

DISPLAY_JOB_HISTORY Command

Purpose Displays entries from the job history log.

Format

DISPLAY_JOB_HISTORY or
DISJH
JOB_NAME=list of name or *keyword*
LOGIN_FAMILY=list of name or *keyword*
TRACE_JOB_CHILDREN=boolean
TRACE_JOB_OUTPUT=boolean
BEGINNING_LOG_POSITION=keyword
SORTED_ORDER=keyword
OUTPUT=file
STATUS=status variable

Parameters

JOB_NAME or *JN*

Specifies the job name for which the job history is to be displayed. You can specify one or more system-supplied or user-supplied job names or the keywords CURRENT or ALL. The default is CURRENT.

LOGIN_FAMILY or *FAMILY_NAME* or *FN* or *LF*

Specifies the login family name of the job for which the job history is to be displayed. You can specify one or more family names or the keywords CURRENT or LOCAL. The default is CURRENT.

TRACE_JOB_CHILDREN or *TJC*

Specifies whether the history of jobs generated by the selected job should be displayed. Specify TRUE or FALSE. The default is FALSE.

TRACE_JOB_OUTPUT or *TJO*

Specifies whether the history of output files generated by the selected jobs should be displayed. Specify TRUE or FALSE. The default is FALSE.

BEGINNING_LOG_POSITION or *BLP*

Specifies the time of the beginning position in the job history log from which entries are to be displayed. Specify one of the following keywords:

BOI (B)

Beginning of information on the log.

SESSION (S)

From the start of the login time of the requesting job.

TODAY (T)

From the start of the current day's entries.

The default is SESSION.

SORTED_ORDER or *SO*

Specifies how the selected events should be sorted for display. You can specify one of the following keywords: TIME, JOB, or FAMILY. The default is FAMILY.

OUTPUT or *O*

Specifies the file to which the display should be written. If omitted, $OUTPUT is assumed.

Remarks

- You can only display the history of jobs for which you are the login or control user.
- The DISPLAY_JOB_HISTORY command displays entries from the job history log of the NOS/VE system on which the command is entered. If you submit a job that will be forwarded to another system for processing, you will not be able to trace the job history of the submitted job after it has been forwarded.
- For more information, see the NOS/VE System Usage manual.

DISPLAY_JOB_LIMIT Command

Purpose Displays your job's limits.

Format

DISPLAY_JOB_LIMIT or
DISPLAY_JOB_LIMITS or
DISJL
OUTPUT=file
STATUS=status variable

Parameters *OUTPUT* or *O*

Specifies the file to which the required information will be written. If omitted, $OUTPUT is assumed.

Remarks

- The SCL function $JOB_LIMIT can also be used to determine your job's limits.
- For more information, see the NOS/VE System Usage manual.

Examples When you enter DISPLAY_JOB_LIMIT, a display of your job limits similar to the following example appears:

```
/display_job_limits
Limit Name      Accumulator    Resource Limit     Abort Limit
----------      -----------    --------------     -----------
CP_TIME                   1         126663748       140737488
SRU                       1         UNLIMITED       UNLIMITED
TASK                      1                10              11
```

DISPLAY_JOB_STATUS Command

Purpose Displays the current status of one or more jobs.

Format

DISPLAY_JOB_STATUS or
DISJS
NAME=list of name or *keyword*
DISPLAY_OPTION=list of keyword
OUTPUT=file
STATUS=status variable

Parameters

NAME or *NAMES* or *JOB_NAME* or *JOB_NAMES* or *JN* or *N*

Specifies the job whose status is to be displayed. If ALL is specified, status of all jobs you own or submit are displayed. Omission causes the status of the requesting job to be displayed.

DISPLAY_OPTION or *DISPLAY_OPTIONS* or *DO*

Specifies what information is displayed for the selected jobs. Keywords are:

ALL

Displays all of the following keyword information.

CONTROL_FAMILY (CF)

Displays the family name of the control user of the job.

CONTROL_USER (CU)

Displays the user name of the control user of the job.

CPU_TIME_USED (CTU)

Displays the CPU time used by the job.

DISPLAY_MESSAGE (DM)

Displays the last command executed or the last display message issued, whichever is most recent.

JOB_CLASS (JC)

Displays the job's job class.

JOB_DESTINATION_USAGE (JDU)

Displays the application used to forward the job to a remote system for execution.

JOB_MODE (JM)

Displays the job mode.

JOB_STATE (JS)

Displays the current state of the job. The following values may be returned:

DEFERRED

Job is not yet eligible to be initiated.

QUEUED

Job is waiting to be initiated.

INITIATED

Job has been initiated.

TERMINATING

Job is terminating.

COMPLETED

Reserved for future use.

LOGIN_FAMILY (LF)

Displays the family name under which the job is scheduled and run.

LOGIN_USER (LU)

Displays the user name under which the job is scheduled and run.

OPERATOR_ACTION_POSTED (OAP)

Displays a boolean indicating whether the job is waiting for operator action. A value of TRUE indicates the job is waiting for operator action. A value of FALSE indicates the job is not waiting for operator action.

PAGE_FAULTS (PF)

Displays the number of page faults caused by the job.

SYSTEM_JOB_NAME (SJN)

Displays the name assigned to the job by the system.

USER_JOB_NAME (UJN)

Displays the job name supplied by the user.

The default is CPU_TIME_USED, DISPLAY_MESSAGE, JOB_CLASS, PAGE_FAULTS, and SYSTEM_JOB_NAME.

OUTPUT or *O*

Identifies the file to which the information is displayed and, optionally, specifies how the file is to be positioned prior to use. Omission causes $OUTPUT to be used.

Remarks

- You can display only the status of jobs for which you are the login user, the control user, or the immediate parent job.
- A system operator can display the status of any job.
- You can obtain the status of jobs you have submitted by using the JOB_NAME parameter.
- The SCL function $JOB_STATUS can be used to determine the status of a job.
- You can also execute this command at any time from an interactive terminal by entering:

```
(ncc) S
```

  where (ncc) is your network command character. For more information, refer to chapter 3 in this manual.

- To display the status of all your jobs at any time from an interactive terminal, enter:

```
(ncc) J
```

  where (ncc) is your network command character. For more information, refer to chapter 3 in this manual.

- For more information, see the NOS/VE System Usage manual.

Examples The following example displays the status of a submitted batch job.

```
/disjs jn=my_job do=all
Control_Family                : nve
Control_User                  : sclqr
CPU_Time_Used                 : Job Mode- 16.137
                                Monitor Mode- 0.365
Display_Message               : forend
Job_Class                     : batch
Job_Destination_Usage         : ve
Job_Mode                      : batch
Job_State                     : initiated
Login_Family                  : nve
Login_User                    : sclqr
Operator_Action_Posted        : no
Page_Faults                   : Assigned- 105
                                From Disk- 67
                                Reclaimed- 23
System_Job_Name               : $0990_0102_aad_2011
User_Job_Name                 : my_job
```

DISPLAY_KEYED_FILE Command

Purpose Formats and displays the contents of a keyed file.

Format

DISPLAY_KEYED_FILE or
DISKF
INPUT=file
OUTPUT=file
FORMAT=keyword
DISPLAY_OPTION=list of keyword
BLOCK_LIST=list of integer
STATUS=status variable

Parameters

INPUT or **I**

File whose contents is to be displayed. You must have at least read permission to the file. This parameter is required.

OUTPUT or *O*

File to which the formatted display is written. If the OUTPUT parameter is omitted, the display is written to file $OUTPUT.

FORMAT or *FORMATS* or *F*

List of one or more keywords indicating the representation used for the contents of records.

ASCII (A)

ASCII characters.

HEXADECIMAL (H)

Hexadecimal digits.

ALL

Both ASCII characters and hexadecimal digits. (No other formats can be specified with ALL.)

If the FORMAT parameter is omitted, the representation used is ASCII.

DISPLAY_OPTION or *DISPLAY_OPTIONS* or *DO*

List of one or more keywords indicating the types of information to be displayed.

MAP (M)

Cross-reference of all blocks

TABLES (T)

Formatted contents of internal tables

INDEX_BLOCKS (IB or I)

Index records

DATA_BLOCKS (DB or D)

Data records

EMPTY_BLOCKS (EB or E)

Block numbers of empty blocks.

ALL (A)

All the preceding options. (No other display options can be specified with ALL.)

The default value depends on whether the BLOCK_LIST parameter is specified.

If the BLOCK_LIST parameter is not specified, the default value is MAP.

If the BLOCK_LIST parameter is specified, the default value is ALL.

BLOCK_LIST or *BL*

Optional list of block numbers indicating the blocks to be displayed. The blocks are displayed in the order specified in the list.

You can specify from 1 through 999 block numbers and ranges of block numbers. Block numbers range from 0 through 4398046511103 ([2**42]-1).

The BLOCK_LIST parameter does not limit the blocks in the MAP cross-reference.

If the BLOCK_LIST parameter is omitted, the command applies to all blocks in the file.

Remarks

- A dump of even a small keyed file produces a very long listing. So it is recommended that you first get a cross-reference listing of the file (DISPLAY_OPTION=MAP) so that you can limit the file dump to only the pertinent information. The parameters that limit the file dump are FORMAT, DISPLAY_OPTIONS, and BLOCK_LIST.

- Do not specify FORMAT=ALL unless you require both ASCII and hexadecimal representation; ALL doubles the number of lines required to list record contents.

- The DISPLAY_OPTIONS parameter specifies the types of information dumped.

- The file specified on the command must be a keyed file and it must exist; otherwise, DISPLAY_KEYED_FILE returns a warning message.

- For more information, see the NOS/VE Advanced File Management Usage manual.

Examples

This command writes a cross-reference of the contents of file $USER.ISFILE on file ISMAP:

```
/display_keyed_file input=$user.isfile output=ismap
```

Assume that using the cross-reference from the previous example, you decide to dump the data records from blocks 6 and 7 and blocks 9 through 15 in ASCII format. To do so, you enter this command:

```
/display_keyed_file input=$user.isfile ..
../output=isdump display_option=data_blocks ..
../block_list=(6,7,9..15)
```

You could then print the listing on file ISDUMP.

DISPLAY_KEYED_FILE_PROPERTIES Command

Purpose Displays properties of a keyed file. The displayed properties can include file attributes, structural properties, and statistics.

Format **DISPLAY_KEYED_FILE_PROPERTIES** or **DISKFP**

FILE=list of any
OUTPUT=file
DISPLAY_OPTION=list of keyword
STATUS=status variable

Parameters

FILE or **INPUT** or **F** or **I**

Keyed file for which properties are to be displayed. You must have at least read permission to the file. This parameter is required.

To specify a nested file, first specify the file reference and then the nested-file name, enclosed in parentheses.

OUTPUT or *O*

File to which the display is written. If the OUTPUT parameter is omitted, the display is written to file $OUTPUT.

DISPLAY_OPTION or *DISPLAY_OPTIONS* or *DO*

List of one or more keywords indicating the property types to be displayed.

FILE_ATTRIBUTES (FA)

File attributes kept for the life of the keyed file.

STATISTICS (S)

Statistics maintained for the keyed file.

STRUCTURAL_PROPERTIES (SP)

Internal organization properties of the keyed file.

ALL (A)

All of the above. (You cannot specify other keywords with ALL.)

If the DISPLAY_OPTIONS parameter is omitted, the display includes the file attributes and structural properties, but not statistics.

Remarks

- The display consists of two or more pages of output.

  The first page lists the properties that pertain to the entire file.

  The second and any subsequent pages list the properties of each nested file in the file and the alternate keys defined for each nested file.

- Unless additional nested files have been created in a file, a file contains only one nested file; it is named $MAIN_FILE.

- For each alternate key, DISPLAY_KEYED_FILE_PROPERTIES lists only those properties defined for the key.

- The file access statistics listed may be inaccurate if the file has been read without modify permission. The reason for this is that when the file is read without modify permission, the statistics for that read cannot be recorded.

- The file specified on the command must be a keyed file and it must exist; otherwise, DISPLAY_KEYED_FILE_PROPERTIES returns a warning message.

- For more information, see the NOS/VE Advanced File Management Usage manual.

Examples

This command lists statistics and structural properties for file $USER.KEYED_FILE on file $USER.LIST:

```
/display_keyed_file_properties ..
../file=$user.keyed_file output=$user.list ..
../display_option=(statistics, structural_properties)
```

This command lists the file attributes and structural properties of file $USER.ISFIL on $OUTPUT. The resulting display is shown:

```
/diskfp $user.isfil
              Display_Keyed_File_Properties          10:31:23
         1984-09-19 NOS/VE Keyed File Utilities 1.1 11917
File = .NVE.USER99.ISFIL

File_attributes and structural_properties at the file level


   Altered_Not_Closed          : no
   Application_Information     : none
   Block_Length "actual"       : 2048 "bytes"
   Error_Exit_Name             : none
   File_Access_Procedure       : none
   File_Content                : UNKNOWN
   File_Limit                  : 100000000 "bytes"
   Forced_Write                : unforced
   Logging_Options             : none
   Log_Residence               : none
   Maximum_Record_Length       : 80 "bytes"
   Minimum_Record_Length       : 50 "bytes"
   Nested_File_Count           : 1
   Open_Position               : $boi
   Permanent                   : yes
   Record_Limit                : 10000
   Ring_Attributes             : (11, 11, 11)
   Ruined_Flag                 : off
   Segment_Information
      Blocks_In_Use            : 2
      Empty_Block_Count        : 0
   Size                        : 4096 "blocks"
   User_Information            : none

              Display_Keyed_File_Properties          10:31:23
         1984-09-19 NOS/VE Keyed File Utilities 1.1 11917

File attributes and structural_properties of $MAIN_FILE

   Block_Count                 : 1
   Creation_Date               : 6/25/84 15:50:14.274
   Data_Padding                : 0 "%"
   Embedded_Key                : yes
   File_Organization           : indexed_sequential
   Index_Levels "current"      : 0
   Index_Level_Overflow        : no
   Index_Padding               : 0 "%"
   Key_Length                  : 5 "bytes"
   Key_Position                : 0
   Key_Type                    : uncollated
   Maximum_Record_Length       : 80 "bytes"
   Minimum_Record_Length       : 5 "bytes"
   Record_Type                 : undefined
   Ruined_Flag                 : off
```

DISPLAY_LINK_ATTRIBUTES Command

Purpose Displays the values set for individual link attributes.

Format

DISPLAY_LINK_ATTRIBUTES or
DISPLAY_LINK_ATTRIBUTE or
DISLA
DISPLAY_OPTIONS=list of keyword
OUTPUT=file
STATUS=status variable

Parameters *DISPLAY_OPTIONS* or *DISPLAY_OPTION* or *DO*

Specifies the link attributes you want displayed. The keywords are:

ALL

Displays all of the following keyword information.

CHARGE (C)

Displays the NOS or NOS/BE charge number.

FAMILY (F)

Displays the NOS family name. If the partner system is NOS/BE, this option has no meaning.

PROJECT (P)

Displays the NOS or NOS/BE project number.

USER (U)

Displays the NOS or NOS/BE user name. In NOS/BE, this parameter specifies the name used to access the system and is the default permanent file id if a file id is not specified on subsequent file transfer commands.

If this parameter is omitted, ALL is used.

OUTPUT or *O*

Specifies the file to which information is to be sent. Omission causes $OUTPUT to be used.

Remarks For more information, see the NOS/VE System Usage manual.

DISPLAY_LOG Command

Purpose Displays the contents of the job log for the requesting job.

Format

DISPLAY_LOG or
DISL
DISPLAY_OPTIONS=integer or *keyword*
OUTPUT=file
STATUS=status variable

Parameters

DISPLAY_OPTIONS or *DISPLAY_OPTION* or *DO*

Specifies the portion of the log that is to be displayed. Options are:

ALL (A)

Display starts at the beginning of the log.

LAST (L)

Display starts with the first message after the last DISPLAY_LOG command.

integer

Displays the last n messages of the log, where n is a positive integer value, and the current log message.

Omission causes ALL to be used.

OUTPUT or *O*

Identifies the file to which the log contents are displayed and, optionally, specifies how the file is positioned prior to use. Omission causes $OUTPUT to be used.

Remarks

- The display includes the time the log entry was made (hh:mm:ss:mmm), message origin (CI if command interpreted, CS if command skipped, PR if program generated, RC if job recovery, or SY if system generated), and the message text.
- The display terminates at the last log entry existing at the time DISPLAY_LOG was entered.

- You can also execute this command at any time from an interactive terminal by entering:

  ```
  (ncc) L
  ```

  Where (ncc) is your network command character. For more information, refer to chapter 3 in this manual.

- For more information, see the NOS/VE System Usage manual.

Examples The following example displays the last seven lines of the job log.

```
/display_log display_option=7
09:38:05.248.CI.set_message_mode brief
09:38:14.094.CI.set_file_attributes data_file
09:38:21.386.CI.copy_file data_file
09:38:21.497.PR. --ERROR-- FSP$OPEN was issued for DATA_FILE.1, which does
  not exist.
09:38:52.930.CI.setmm full
09:39:00.656.CI.copy_file data_file
09:39:00.686.PR. --ERROR AM 1016-- FSP$OPEN_FILE was issued for file,
  :$LOCAL.DATA_FILE.1,
09:39:00.686.CI.which does not exist.
09:39:59.427.CI.display_log display_option=7
```

The following example displays the log starting with the first message after the previous DISPLAY_LOG command.

```
/"Comments appear in the job log.
/"These are inserted to provide entries for this example.
/display_log do=last
09:40:20.034.CI."Comments appear in the job log.
09:41:05.444.CI."These are inserted to provide entries for
  this example.
09:41:38.007.CI.display_log do=last
```

The following example displays the entire job log on file LOG_OUTPUT.

```
/display_log o=log_output
```

DISPLAY_MESSAGE Command

Purpose Displays a message in one or more logs.

Format

DISPLAY_MESSAGE or
DISM
MESSAGE=string
TO=list of keyword
STATUS=status variable

Parameters

MESSAGE or **M**

Specifies the message text string to be placed in the log. This parameter is required.

TO or *T*

Specifies the destination(s) to which the message is to be directed. Options are:

SYSTEM

Sends a message to the system log.

STATISTIC

Places the message in the statistics log.

HISTORY

Places the message in the job history log. You can view the message with the DISPLAY_JOB_HISTORY command.

JOB_MESSAGE

Sends a message to your job status message area. You can view the message with the DISPLAY_JOB_STATUS command.

ACCOUNT

Sends a message to the account log.

ENGINEERING

Sends a message to the engineering log.

JOB_STATISTIC

Sends a message to the job statistics log.

JOB

Sends a message to the job log. You can view the message with the DISPLAY_LOG command.

ALL

Selects all of the TO parameter options.

Omission causes the option JOB to be used.

Remarks

- Only the system operator can enter messages into the system, account, engineering, or statistics log.
- You can enter messages into the job statistic log, the history log, the job status message area, and job log associated with your job.
- For more information, see the NOS/VE System Usage manual.

Examples

The following example places a message in the job log and displays that portion of the job log.

```
/display_message 'Log message for job log.'
/display_log do=2
10:07:20.036.CI.display_message 'Log message for job log.'
10:07:41.879.PR.Log message for job log.
10:07:55.654.CI.display_log do=2
```

The following example uses the MESSAGE and TO parameters to place a second message in the job log.

```
/dism 'Second log message.' to=job
/disl 2
10:08:43.089.CI.dism 'Second log message.' to=job
10:08:43.107.PR.Second log message.
10:08:47.219.CI.disl 2
```

The following example places a message in the history log.

```
/dism 'Accounting jobs initiated.' to=history
```

DISPLAY_OBJECT_LIBRARY Command

Purpose

Displays information about an object library, object file, or procedure file.

Format

DISPLAY_OBJECT_LIBRARY or
DISOL
LIBRARY=file
MODULE=list of range of any
DISPLAY_OPTION=list of keyword
OUTPUT=file
ALPHABETICAL_ORDER=boolean
STATUS=status variable

Parameters

LIBRARY or **L**

Object library, object file, or procedure file about which information is displayed. This parameter is required.

MODULE or *MODULES* or *M*

List of modules whose information is to be displayed.

Use a string value for a module whose name is not an SCL name. Some examples of such module names are: a COBOL module, where a hyphen character (-) may be part of the name, and a C function, where lower case is significant.

If MODULE is omitted, information about all modules in the file or library is displayed.

DISPLAY_OPTION or *DISPLAY_OPTIONS* or *DO*

Set of one or more keywords indicating the information displayed in addition to the module type and name. (The module types are load module, object module, bound module, message module, SCL procedure, and program description.) Options are:

NONE

No information other than the module type and name.

DATE_TIME (DT)

Creation date and time.

ENTRY_POINT (EP)

Entry point names.

HEADER (H)

Module header information. This includes the:

- Module type, name, creation date and time, kind, generator, generator name version, and comments.
- Formal parameters availability, scope and log option for SCL command procedures.
- Entire program description, availability, application identifier, and scope and log option for program description.

- Natural language and online manual name for message modules.
- The lowest and highest condition codes for message modules that contain status message information.

LIBRARIES or LIBRARY (L)

Local file names within the object text of the modules added to the program library list when the module is loaded.

REFERENCE (R)

External references.

COMPONENT (C)

Module headers of the component modules if the module is a bound module.

ALL

All of the options listed for the DISPLAY_OPTIONS parameter.

If DISPLAY_OPTION is omitted, DATE_TIME is used.

OUTPUT or *O*

Output file. This file can be positioned. If OUTPUT is omitted, file $OUTPUT is used.

ALPHABETICAL_ORDER or *AO*

Indicates the display order for the module information. Options are:

TRUE

Alphabetical order by module name.

FALSE

Order in which the modules exist on the file.

If ALPHABETICAL_ORDER is omitted, FALSE is used.

Remarks

- If you do not want to display information for all modules in the file, you can limit the display by specifying module names or module ranges on the MODULE parameter.
- For more information, see the NOS/VE Object Code Management manual.

Examples See the NOS/VE Object Code Management manual for a detailed example.

DISPLAY_OBJECT_TEXT Command

Purpose Displays the internal object records of an object library or file.

Format

DISPLAY_OBJECT_TEXT or
DISOT
FILE=file
OUTPUT=file
DISPLAY_HEX_RECORDS=boolean
STATUS=status variable

Parameters

FILE or *F*

Object library or object file whose object records are to be displayed. If FILE is omitted, file LGO is used.

OUTPUT or *O*

Output file. This file can be positioned. If OUTPUT is omitted, file $OUTPUT is used.

DISPLAY_HEX_RECORDS or *DHR*

Specifies whether large hexadecimal fields should be displayed. (A code section is an example of a large hexadecimal field.) If DISPLAY_HEX_RECORDS is not specified, large hexadecimal fields are displayed.

Remarks

- The first line of each object record pair displayed contains information found in the object text descriptor. This includes the record number and byte offset of the object text descriptor, and the kind and adaptable size of the next object record. The actual object record is then displayed on subsequent lines.

- For more information, see the NOS/VE Object Code Management manual.

DISPLAY_OUTPUT_ATTRIBUTE Command

Purpose Displays attributes of an output file.

Format

DISPLAY_OUTPUT_ATTRIBUTE or
DISPLAY_OUTPUT_ATTRIBUTES or
DISOA
NAME=name
DISPLAY_OPTION=list of keyword
OUTPUT=file
STATUS=status variable

Parameters

NAME or **N**

Specifies the output file whose attributes are to be displayed. Enter either the system-supplied or user-supplied name.

DISPLAY_OPTION or *DISPLAY_OPTIONS* or *DO*

Specifies the information you want to display. The keywords are:

ALL

Displays all of the following keyword information.

COMMENT_BANNER (CB)

Displays the COMMENT_BANNER attribute.

CONTROL_FAMILY (CF)

Displays the family name of the control user.

CONTROL_USER (CU)

Displays the user name of the control user.

COPIES (C)

Displays the COPIES attribute.

COPIES_PRINTED (CP)

Displays the number of copies that have been printed, if any.

DATA_MODE (DM)

Displays the DATA_MODE attribute.

DEVICE (D)

Displays a name that, when combined with the STATION name, identifies a particular printer.

DEVICE_TYPE (DT)

Displays the type of output device. Currently, this value is always a printer.

EARLIEST_PRINT_TIME (EPT)

Displays the EARLIEST_PRINT_TIME attribute.

EXTERNAL_CHARACTERISTICS (EC)

Displays the EXTERNAL_CHARACTERISTICS attribute.

FILE_POSITION (FP)

Displays a restarting point for the output file if the output is interrupted. The FILE_POSITION value is always zero (BOI).

FILE_SIZE (FS)

Displays the size of the output file in bytes.

FORMS_CODE (FC)

Displays the FORMS_CODE attribute.

LATEST_PRINT_TIME (LPT)

Displays the LATEST_PRINT_TIME attribute.

LOGIN_ACCOUNT (LA)

Displays the account name of the job generating the output file.

LOGIN_FAMILY (LF)

Displays the family name of the job generating the output file.

LOGIN_PROJECT (LP)

Displays the project name of the job generating the output file.

LOGIN_USER (LU)

Displays the user name of the job generating the output file.

OPERATOR_FAMILY (DESTINATION_FAMILY or OF or DF)

Displays the OPERATOR_FAMILY attribute.

OPERATOR_USER (STATION_OPERATOR or OU or SO)

Displays the OPERATOR_USER attribute.

ORIGINATING_APPLICATION_NAME (OAN)

Displays the name of the application that entered the job that generated the output file into the system.

OUTPUT_CLASS (OC)

Displays the OUTPUT_CLASS attribute.

OUTPUT_DESTINATION (ODE)

Displays the OUTPUT_DESTINATION attribute.

OUTPUT_DESTINATION_USAGE (DESTINATION_USAGE or DU or ODU)

Displays the OUTPUT_DESTINATION_USAGE attribute.

OUTPUT_PRIORITY (OP)

Displays the OUTPUT_PRIORITY attribute.

OUTPUT_SUBMISSION_TIME (QUEUED_TIME or QT or OST)

Displays the time when the output file was released from the job into the queue.

PURGE_DELAY (PD)

Reserved for future use.

REMOTE_HOST_DIRECTIVE (DUAL_STATE_ROUTE_PARAMETERS or RHD or DSRP)

Displays the REMOTE_HOST_DIRECTIVE attribute.

ROUTING_BANNER (RB)

Displays the ROUTING_BANNER attribute.

SITE_INFORMATION (SI)

Displays the SITE_INFORMATION string associated with the job that generated the output file.

STATION (S)

Displays the STATION attribute.

SYSTEM_FILE_NAME (SFN)

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 name created is unique (no other name on the network is the same).

SYSTEM_JOB_NAME (SJN)

Displays the system-supplied name of the job that generated the output file. This name is created by the NOS/VE system from which the job was submitted. This name is unique (no other job in the network will have the same name).

USER_FILE_NAME (UFN)

Displays the user-supplied name of the output file. If no name is specified, the file name is used.

USER_INFORMATION (UI)

Displays the user information string associated with the job from which the output is generated.

USER_JOB_NAME (UJN)

Displays the user-supplied name of the job from which the output file is generated.

VERTICAL_PRINT_DENSITY (VPD)

Displays the VERTICAL_PRINT_DENSITY attribute.

VFU_LOAD_PROCEDURE (VLP)

Displays the VFU_LOAD_PROCEDURE attribute.

OUTPUT or *O*

Specifies the file to which the requested information will be written. If omitted, $OUTPUT is assumed.

Remarks

- The SCL function $JOB_OUTPUT can be used to determine attribute information about an output file.

- For more information, see the NOS/VE System Usage manual.

Examples

The following example display shows the output attributes for file EXAMPLES:

```
/display_output_attributes name=examples do=all
Comment_Banner                : 'EXAMPLES'
Control_Family                : nve
Control_User                  : sarett
Copies                        : 1
Copies_Printed                : 0
Data_Mode                     : coded
Device                        : automatic
Device_Type                   : printer
Earliest_Print_Time           : none
External_Characteristics      : 'NORMAL'
File_Position                 : 0
File_Size                     : 6481
Forms_Code                    : 'NORMAL'
Latest_Print_Time             : none
Login_Account                 : d1257
Login_Family                  : nve
Login_Project                 : p83a2821
Login_User                    : sarett
Operator_Family               : nve
Operator_User                 : sarett
Originating_Application_Name  : osa$dual_state_interactive
Output_Class                  : normal
Output_Destination            : 'NVE'
Output_Destination_Usage      : dual_state
Output_Priority               : low
Output_Submission_Time        : 1987-07-31.16:09:35
Purge_Delay                   : none
Remote_Host_Directive         : ' '
Routing_Banner                : 'SARETT'
Site_Information              : ' '
Station                       : automatic
System_File_Name              : $0990_0102_aad_1511
System_Job_Name               : $0990_0102_aad_1367
User_File_Name                : examples
User_Information              : ' '
User_Job_Name                 : sarett
Vertical_Print_Density        : six
VFU_Load_Procedure            : none
```

DISPLAY_OUTPUT_HISTORY Command

Purpose Displays output file entries from the job history log.

Format

DISPLAY_OUTPUT_HISTORY or
DISOH
OUTPUT_FILE_NAME=list of name or *keyword*
JOB_NAME=list of name or *keyword*
LOGIN_FAMILY=list of name or *keyword*
BEGINNING_LOG_POSITION=keyword
SORTED_ORDER=keyword
OUTPUT=file
STATUS=status variable

Parameters

OUTPUT_FILE_NAME or *OFN*

Specifies the output file name for which the output history is to be displayed. You can specify a list of file names or the keyword ALL. The default is ALL.

JOB_NAME or *JN*

Specifies the job name for which the output history is to be displayed. You can supply a list of system-supplied or user-supplied job names or the keywords CURRENT or ALL. The default is CURRENT.

LOGIN_FAMILY or *FAMILY_NAME* or *FN* or *LF*

Specifies the login family name of the job for which the output history is displayed. You can specify a list of family names or the keywords CURRENT or LOCAL. The default is CURRENT.

BEGINNING_LOG_POSITION or *BLP*

Specifies the time of the beginning position in the job history log from which entries are to be displayed. Specify one of the following keywords:

BOI (B)

Beginning of information on the log.

SESSION (S)

Beginning log position starting with the login time of the requesting job.

TODAY (T)

Start of the current day's entries.

The default is SESSION.

SORTED_ORDER or *SO*

Specifies how the selected events should be sorted for display. You can specify one of the following keywords: TIME, JOB, or FAMILY. The default is FAMILY.

OUTPUT or *O*

Specifies the file to which the display should be written. If omitted, $OUTPUT is assumed.

Remarks

- You can only display the history of output files from jobs for which you are the login or control user.
- The DISPLAY_OUTPUT_HISTORY command displays entries from the job history log of the NOS/VE system on which the command is entered. If you route an output file to another system for processing, you will not be able to trace the output history of the file after it has been forwarded.
- For more information, see the NOS/VE System Usage manual.

DISPLAY_OUTPUT_STATUS Command

Purpose Displays information about the current status of an output file from the NOS/VE output queue.

Format

DISPLAY_OUTPUT_STATUS or
DISOS or
DISPLAY_PRINT_STATUS or
DISPS
NAME=list of name or *keyword*
DISPLAY_OPTION=list of keyword
OUTPUT=file
STATUS=status variable

Parameters *NAME* or *NAMES* or *N*

Specifies the output file(s) whose status will be displayed.

Values can be the system-supplied file name, the user-supplied file name, or the keyword ALL. ALL specifies that you want to display the status of all your output files. If this parameter is omitted, ALL is assumed.

DISPLAY_OPTION or *DISPLAY_OPTIONS* or *DO*

Specifies what information is displayed for the selected output files. Keywords are:

ALL

Displays all of the following keyword information.

CONTROL_FAMILY (CF)

Displays the family name of the control user of the output files.

CONTROL_USER (CU)

Displays the user name of the control user of the output file.

LOGIN_FAMILY (LF)

Displays the login family name of the job that generated the output file.

LOGIN_USER (LU)

Displays the login user name of the job that generated the output file.

OUTPUT_DESTINATION_USAGE (ODU)

Displays the OUTPUT_DESTINATION_USAGE attribute.

OUTPUT_STATE (OS)

Displays the state of the output file. The following values may be returned:

DEFERRED

File is not yet eligible for printing.

QUEUED

File is waiting to be printed.

INITIATED

File is being printed.

TERMINATING

Printing process for the file is terminating.

SYSTEM_FILE_NAME (SFN)

Displays the system-supplied name of the output file. This file name is generated by the NOS/VE system that executes the PRINT_FILE command. This name is unique and identifies the file for the lifetime of the file.

SYSTEM_JOB_NAME (SJN)

Displays the system-supplied name of the job that generated the output file. This name is generated by the system that first queues the input file, is unique, stays intact for the lifetime of the job, and is passed on to all of the job's output files.

USER_FILE_NAME (UFN)

Displays the user-supplied name of the output file. If no name is specified, the name of the file is used.

The default is OUTPUT_STATE, SYSTEM_FILE_NAME, and USER_FILE_NAME.

OUTPUT or *O*

Specifies the file to which the requested information is written. If omitted, $OUTPUT is assumed.

Remarks

- You can only display the status of output files for which you are the login user or the control user.
- A system operator can display the status of any output file
- The SCL function $OUTPUT_STATUS can be used to determine the current status of an output file.
- For more information, see the NOS/VE System Usage manual.

Examples The following sample display shows the output status for user-supplied name EXAMPLES:

```
/display_output_status name=examples do=all
Control_Family                  : nve
Control_User                    : sarett
Login_Family                    : nve
Login_User                      : sarett
Output_Destination_Usage        : dual_state
Output_State                    : printing
System_File_Name                : $0990_0102_aad_1516
System_Job_Name                 : $0990_0102_aad_1367
User_File_Name                  : examples
```

DISPLAY_PROGRAM_ATTRIBUTES Command

Purpose Displays the current job library list and default execution option values for programs executed subsequently within the job.

Format

DISPLAY_PROGRAM_ATTRIBUTES or
DISPLAY_PROGRAM_ATTRIBUTE or
DISPA
OUTPUT=file
STATUS=status variable

Parameters *OUTPUT* or *O*

File on which the display is written. This file can be positioned. If OUTPUT is omitted, file $OUTPUT is used.

Remarks

- You can change the job library list and default execution options with a SET_PROGRAM_ATTRIBUTES command.
- For more information, see the NOS/VE Object Code Management manual.

Examples The following command displays the current job library list and default execution options.

```
/display_program_attributes
Libraries                          : :$system.$system.aam.aaf$4dd_library.105
Debug_Libraries                    : :$system.$system.debug.bound_product.4
Load_Map                           : :$local.loadmap
Load_Map_Options                   : none
Termination_Error_Level            : error
Preset_Value                       : zero
Maximum_Stack_Size                 : 2002944
Debug_Ring                         : 11
Debug_Input                        : :$local.command.1
Debug_Output                       : :$local.$output.1
Abort_File                         : :$local.$null.1
Debug_Mode                         : off
Arithmetic_Overflow                : on
Arithmetic_Loss_of_Significance    : on
Divide_Fault                       : on
Exponent_Overflow                  : on
Exponent_Underflow                 : on
Fp_Indefinite                      : on
Fp_Loss_of_Significance            : off
Invalid_BDP_Data                   : on
```

You can change each listed attribute except the maximum stack size, debug libraries and debug ring attributes using a SET_PROGRAM_ATTRIBUTES command. The debug libraries attribute lists the files containing the debug object libraries. The list can be changed through the SET_DEBUG_LIST command. The debug ring attribute lists the ring in which Debug executes. The Debug ring can be changed with the SET_DEBUG_RING command.

DISPLAY_REMOTE_VALIDATION Command

Purpose Displays the locations for which you created remote validations using the CREATE_REMOTE_VALIDATION command.

Format **DISPLAY_REMOTE_VALIDATION** or **DISRV**

LOCATION=list of name or *keyword*
OUTPUT=file
STATUS=status variable

Parameters

LOCATION or *L*

Specifies the name(s) of the remote location(s) to be displayed. You can specify a family, a list of families, or the keyword ALL. ALL specifies that the names of all the locations for which there are current remote validations will displayed.

If omitted, ALL is assumed.

OUTPUT or *O*

Specifies the file to which the validation information is written. This file must be assigned to an interactive terminal. If omitted, OUTPUT is assumed. You may not specify a remote file with this parameter.

Remarks

- The $REMOTE_VALIDATION function can also be used to determine if validation has been defined for a system.
- For more information, see the NOS/VE System Usage manual.

Examples

The following example displays location information for location SKY:

```
/display_remote_validation l=sky
LOCATION:  sky
/
```

DISPLAY_TAPE_LABEL_ATTRIBUTES Command

Purpose

Displays the current tape label attributes defined for an ANSI labeled magnetic tape file.

Format

DISPLAY_TAPE_LABEL_ATTRIBUTES or
DISPLAY_TAPE_LABEL_ATTRIBUTE or
DISTLA
FILE=file
DISPLAY_OPTIONS=list of name
OUTPUT=file
STATUS=status variable

Parameters

FILE or **F**

Specifies the name of the file for which tape label attribute information is to be displayed. This parameter is required.

DISPLAY_OPTIONS or *DISPLAY_OPTION* or *DO*

Specifies the tape label attribute information to be displayed. Values are:

BLOCK_TYPE (BT)
BUFFER_OFFSET (BO)
CHARACTER_CONVERSION (CC)
CHARACTER_SET (CS)
CREATION_DATE (CD)
CURRENT_FILE (CF)
EXPIRATION_DATE (ED)
FILE_ACCESSIBILITY_CODE (FAC)
FILE_IDENTIFIER (FI)
FILE_SEQUENCE_NUMBER (FSN)
FILE_SET_IDENTIFIER (FSI)
FILE_SET_POSITION (FSP)
GENERATION_NUMBER (GN)
GENERATION_VERSION_NUMBER (GVN)
MAXIMUM_BLOCK_LENGTH (MAXBL)
MAXIMUM_RECORD_LENGTH (MAXRL)
NEXT_FILE (NF)
PADDING_CHARACTER (PC)
RECORD_TYPE (RT)
REWRITE_LABELS (RL)
SOURCE (S)
ALL

If omitted, ALL is assumed. See the CHANGE_TAPE_LABEL_ATTRIBUTES command for information on each display option.

OUTPUT or *O*

Specifies a file to which the information will be written. You can position this file prior to use. If omitted, $OUTPUT is assumed.

Remarks

- If the file is not currently open when you enter this command, the values specified on the most recent CHANGE_TAPE_LABEL_ATTRIBUTES command are displayed. If the file has not been opened by your job and you have not entered a CHANGE_TAPE_LABEL_ATTRIBUTES command, the default values are displayed.
- Errors occur if the NOS/VE file has not been assigned to your job with a REQUEST_MAGNETIC_TAPE command.
- For more information, see the NOS/VE System Usage manual.

DISPLAY_TASK_STATUS Command

Purpose Displays the current status of one or more named tasks.

Format

DISPLAY_TASK_STATUS or
DISTS
TASK_NAME=list of name or **keyword**
OUTPUT=file
STATUS=status variable

Parameters

TASK_NAME or **TASK_NAMES** or **TN**

Specifies the name of the task whose status is to be displayed. This may be the names you supply or the keyword ALL. If ALL is specified, the status of all tasks initiated by the requesting group of synchronously executing tasks is displayed. This parameter is required.

OUTPUT or *O*

Specifies the file to which the display should be written. Omission of the OUTPUT parameter causes $OUTPUT to be used.

Remarks

- The SCL function $TASK_STATUS can also be used to determine the status of a task.
- For more information, see the NOS/VE System Usage manual.

DISPLAY_TERMINAL_ATTRIBUTES Command

Purpose Displays the attributes and their corresponding values for the interactive terminal currently in use.

Format

DISPLAY_TERMINAL_ATTRIBUTES or
DISPLAY_TERMINAL_ATTRIBUTE or
DISTA
DISPLAY_OPTIONS=list of name
OUTPUT=file
STATUS=status variable

Parameters

DISPLAY_OPTIONS or *DISPLAY_OPTION* or *DO*

Specifies the names of the terminal attributes to be displayed.

Refer to the CHANGE_TERMINAL_ATTRIBUTE command for descriptions of the attributes.

Omission or the keyword ALL causes all attributes to be displayed.

OUTPUT or *O*

Identifies the file to which the information is to be displayed and, optionally, specifies how the file is to be positioned. Omission causes $OUTPUT to be used.

Remarks

- The SCL function $TERMINAL_MODEL can be used to determine the value of the TERMINAL_MODEL terminal attribute.

- For more information, see the NOS/VE System Usage manual.

Examples 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"
```

The following is an example of the default terminal attributes for the DEC_VT220 terminal model.

```
/chata tm=dec_vt220
/display_terminal_attributes
Attention_Character             : $CHAR(0)     "NUL"
Backspace_Character             : $CHAR(8)     "BS"
Begin_Line_Character            : $CHAR(0)     "NUL"
Cancel_Line_Character           : $CHAR(24)    "CAN"
Carriage_Return_Delay           : 0
Carriage_Return_Sequence        : CR
Character_Flow_Control          : on
Code_Set                        : ascii
Echoplex                        : off
End_Line_Character              : $CHAR(13)    "CR"
End_Line_Positioning            : lfs
End_Output_Sequence             :
End_Page_Action                 : none
End_Partial_Character           : $CHAR(10)    "LF"
End_Partial_Positioning         : crs
Fold_Line                       : off
Form_Feed_Delay                 : 0
Form_Feed_Sequence              : FF
Hold_Page                       : off
Hold_Page_Over                  : on
Line_Feed_Delay                 : 0
Line_Feed_Sequence              : LF
Network_Command_Character       : $CHAR(37)    "%"
Page_Length                     : 30
Page_Width                      : 80
Parity                          : even
Status_Action                   : send
Terminal_Model                  : DEC_VT220
```

DISPLAY_TERM_CONN_DEFAULTS Command

Purpose Displays the connection attribute defaults for a terminal connection.

Format

DISPLAY_TERM_CONN_DEFAULTS or
DISPLAY_TERM_CONN_DEFAULT or
DISTCD

DISPLAY_OPTIONS=list of name
OUTPUT=file
STATUS=status variable

Parameters *DISPLAY_OPTIONS* or *DISPLAY_OPTION* or *DO*

Specifies the names of the default attributes to be displayed.

For descriptions of the attributes, see the CHANGE_TERM_CONN_DEFAULTS (CHATCD) command.

Omission or the keyword ALL causes all attributes to be displayed.

OUTPUT or *O*

Specifies the name of the file on which the information is to be displayed, and, optionally, specifies how the file is to be positioned prior to use. Omission causes $OUTPUT to be used.

Remarks For more information, see the NOS/VE System Usage manual.

DISPLAY_VALUE Command

Purpose Displays the value of one or more expressions.

Format

DISPLAY_VALUE or
DISPLAY_VALUES or
DISV
VALUES=list of any
OUTPUT=file
STATUS=status variable

Parameters **VALUES** or **VALUE** or **V**

Specifies one or more expressions. When you specify more than one value in the list, the system displays each on a separate line. This parameter is required.

OUTPUT or *O*

Specifies the file to which the information is to be written. The default file is $OUTPUT.

Remarks

- When the system returns an integer with a radix, the radix matches the radix of the first operand of the expression.

- Exercise care when using the DISPLAY_VALUE command when the file specified by the OUTPUT parameter is not connected to the terminal (such as file OUTPUT or $OUTPUT in a batch job). If you create a file using this command's OUTPUT parameter, the DISPLAY_VALUE command gives the PAGE_FORMAT file attribute of BURSTABLE if the file is assigned to mass storage. If the file is assigned to a terminal, the file is created with a PAGE_FORMAT file attribute of CONTINUOUS. Thus, for mass storage files, each time DISPLAY_VALUE is executed, a line printer page eject and page title may result.

  To avoid page ejects and page titles, set the PAGE_FORMAT ATTRIBUTE to CONTINUOUS and the FILE_CONTENTS attribute to LEGIBLE. Do this by including the following command before using the DISPLAY_FILE command on a mass storage file (assuming an OUTPUT file named LIST_FILE):

```
/set_file_attribute file=list_file ..
../page_format=continuous ..
../file_contents=legible
```

  For more information about file attributes, see the NOS/VE System Usage manual.

- If you supply a file reference as the argument for this function, the file path returned depends on the current message mode. If the current message mode if FULL, the complete file path is returned. If the current message mode is BRIEF, the file path relative to the working catalog is returned.

  You can use the SET_MESSAGE_MODE command to change the current message mode.

- For more information, see the NOS/VE System Usage manual.

Examples The following examples demonstrate use of the command with several value kinds:

```
/display_value 2**7-4
124

/create_variable name=stat kind=status
/stat.normal = false
/stat.identifier = 'us'
/stat.condition = 330002
/stat.text = '?Text of Status Variable.'
/display_value stat
--ERROR--CC=us 330002 TEXT=?Text of status
 variable.

/display_value ('Line 1','Line 2','Line 3')
Line 1
Line 2
Line 3

/display_value $variable(stat,kind)
STATUS

/display_value ($max_integer,$min_integer)
9223372036854775807
-9223372036854775807
```

DISPLAY_VARIABLE_LIST Command

Purpose Displays variables accessible to the current block.

Format

DISPLAY_VARIABLE_LIST or
DISVL
OUTPUT=file
STATUS=status variable

Parameters *OUTPUT* or *O*

Specifies the file you want to contain the list of variables. The default file is $OUTPUT.

Remarks

- The most recently created variable appears at the top of the list.
- You can use the $VARIABLE function to return attribute information about a specified variable.
- For more information, see the NOS/VE System Usage manual.

Examples

The following is a sample display of variables. 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 system variable created by the system in the job block at the beginning of the job.

DMACT
IM/DM Command

Purpose

Creates reports showing how the database is used.

Format

DMACT
FILE=file
LIST=file
REPORT=list of name
BEGIN_DATE=integer
END_DATE=integer
STATUS=status variable

Parameters

FILE or *F*

Specifies the DMACT input file. This file is the statistics listing specified for the STATFILE parameter of the DMJ command.

Omission of the FILE parameter causes DMSTATS to be used.

LIST or *L*

Specifies the output file to contain the selected report or reports.

Omission of the LIST parameter causes DMSTOUT to be used.

REPORT or *REPORTS* or *R*

Specifies the DMACT report or reports to generate. Choose from the following reports:

TRANSACTION_ACTIVITY
PROGRAM_ACTIVITY
VIEW_ACTIVITY
RECORD_ACTIVITY
RESOURCE_LOCKS
PRIVACY_FAILURES
SESSION_LOG
USER_ACTIVITY
HOURLY_ACTIVITY
ELEMENT_USAGE

To generate multiple reports, separate the report names by commas and enclose the names in parentheses. For example, to generate the TRANSACTION_REPORT, SESSION_LOG, and USER_ACTIVITY reports, enter the following:

```
reports=(transaction_report,session_log,user_activity)
```

To generate all ten reports, specify ALL or omit this parameter.

BEGIN_DATE or *BD*

Specifies the date you want DMACT to begin compiling the report or reports. Use the YYMMDD format when specifying a date.

Omission of the BEGIN_DATE parameter causes the first date listed in the input file (the statistics listing file) to be used.

END_DATE or *ED*

Specifies the date you want DMACT to stop compiling the report or reports. Use the YYMMDD format when specifying a date.

Remarks

- The most recently created variable appears at the top of the list.
- You can use the $VARIABLE function to return attribute information about a specified variable.
- For more information, see the NOS/VE System Usage manual.

Examples

The following is a sample display of variables. 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 system variable created by the system in the job block at the beginning of the job.

DMACT
IM/DM Command

Purpose

Creates reports showing how the database is used.

Format

DMACT
FILE=file
LIST=file
REPORT=list of name
BEGIN_DATE=integer
END_DATE=integer
STATUS=status variable

Parameters

FILE or *F*

Specifies the DMACT input file. This file is the statistics listing specified for the STATFILE parameter of the DMJ command.

Omission of the FILE parameter causes DMSTATS to be used.

LIST or *L*

Specifies the output file to contain the selected report or reports.

Omission of the LIST parameter causes DMSTOUT to be used.

REPORT or *REPORTS* or *R*

Specifies the DMACT report or reports to generate. Choose from the following reports:

TRANSACTION_ACTIVITY
PROGRAM_ACTIVITY
VIEW_ACTIVITY
RECORD_ACTIVITY
RESOURCE_LOCKS
PRIVACY_FAILURES
SESSION_LOG
USER_ACTIVITY
HOURLY_ACTIVITY
ELEMENT_USAGE

To generate multiple reports, separate the report names by commas and enclose the names in parentheses. For example, to generate the TRANSACTION_REPORT, SESSION_LOG, and USER_ACTIVITY reports, enter the following:

```
reports=(transaction_report,session_log,user_activity)
```

To generate all ten reports, specify ALL or omit this parameter.

BEGIN_DATE or *BD*

Specifies the date you want DMACT to begin compiling the report or reports. Use the YYMMDD format when specifying a date.

Omission of the BEGIN_DATE parameter causes the first date listed in the input file (the statistics listing file) to be used.

END_DATE or *ED*

Specifies the date you want DMACT to stop compiling the report or reports. Use the YYMMDD format when specifying a date.

Omission of the END_DATE parameter causes the last date listed in the input file (the statistics listing file) to be used.

Remarks For more information, see the DM Utilities Reference Manual for DM on CDC NOS/VE.

DMBR
IM/DM Command

Purpose Executes the DM Backup and Restore module, allowing the data administrator to backup and restore files.

Format

DMBR
UID=dm_name
UPW=dm_name
DB=name
ACTION=keyword
FILE=list of range of dm_integer or *keyword*
JOURNAL=list of dm_star or *keyword*
BSN=list of range of dm_integer or *keyword*
SET=list of dm_integer or *keyword*
TOCBSN=list of range of dm_integer or *keyword*
VERSION=integer
GET=dm_file_descriptor
TOC=dm_file_descriptor
JOB=dm_file_descriptor
JOB_HEADER=dm_file_descriptor
SUBMIT=keyword
WAIT=keyword
SUBACTION=keyword
INPUT=dm_file_descriptor
OUTPUT=dm_file_descriptor
VF=dm_file_descriptor
STATUS=status variable

Parameters *UID*

User identification code (only first 8 characters are used).

IM/DM prompts you for this value if it is not specified.

UPW

User password (only the first 8 characters are used).

IM/DM prompts you for this value if it is not specified.

DB

Name of the database.

IM/DM prompts you for this value if it is not specified.

ACTION

Action performed:

AUDIT

Lists the file sequence number, time span covered by the file, last backup time, and last backup set used for a DM journal file or data file.

BACKUP

Stores backup copies of data files and/or journal files in a DM backup set.

FREE

Releases a DM backup set.

HISTORY

Lists the backup history for the specified data and journal files.

LIST

Lists the contents of the specified DM backup sets.

RESET

Resets a journal file to the empty condition after it has been successfully backed up using another utility (not DMBR).

RESTORE

Restores a copy of a data file or journal file from a DM backup set.

STATUS

Lists the file status of each occurrence database file.

IM/DM prompts you for this value if it is not specified.

FILE or *FILES*

When ACTION=BACKUP, this parameter lists the data files to be copied to a backup set.

When ACTION=HISTORY, this parameter specifies the file on which the data file histories are written.

When ACTION=RESTORE, this parameter specifies the data files to be restored.

Data files can be specified by number, a range of numbers from 1 to 63, or by one of the following:

DDB

The definition database

ALL or *

All data files

NONE

No data files

If FILE is omitted, NONE is used.

JOURNAL or *JOURNALS*

Journal files on which the action (BACKUP, HISTORY, RESET, or RESTORE) is performed. You can specify up to 6 files. Options include:

A

Primary journal file A

B

Primary journal file B

ACOPY

Copy of journal file A

BCOPY

Copy of journal file B

ADDB

Definition database journal file A

BDDB

Definition database journal file B

ALL or *

All journal files

NONE

No journal files

If JOURNAL is omitted, NONE is used.

BSN

Backup sets on which the action (FREE or LIST) is performed. It can specify a number, a range of numbers (from 1 to 250), or the keyword ALL or * for all backup sets.

SET or *SETS*

File sets to be copied to the backup set when ACTION=BACKUP. It can be an integer, the keyword ALL or * for all file sets, or the keyword NONE for no file sets.

If SET is omitted, NONE is used.

TOCBSN

Backup file set from which a TOC file is restored. It can be an integer, a range of integers from 1 to 250, or the keyword ALL or * for all backup file sets.

VERSION

Version number of the default database.

If VERSION is omitted, 0 is used.

GET

File to audit when ACTION=AUDIT.

TOC

When ACTION=BACKUP, this parameter specifies the file to which the table of contents is written.

When ACTION=RESTORE, this parameter specifies the file descriptor skeleton for the table of contents files.

If TOC is omitted, BSN???_TOC is used where ??? is replaced by the backup set number.

JOB

File on which the job statements are written when ACTION=BACKUP or RESTORE.

If JOB is omitted, file DMBR_JOB is used.

JOB_HEADER

File containing the statements that are to precede the statements in the job file.

This parameter is required when DMBR is submitted in batch mode.

SUBMIT

Indicates whether the JOB file should be submitted as a batch job.

YES

Submit as a batch job.

NO

Do not submit as a batch job.

If SUBMIT is omitted, YES is used.

WAIT

Indicates whether the job should wait for files that are in use.

YES

Waits for files.

NO

Does not wait for files.

If WAIT is omitted, YES is used.

SUBACTION

Reserved.

INPUT

File from which any missing required parameters are read.

If INPUT is omitted, file INPUT is used.

OUTPUT

File to which all messages produced are written.

If OUTPUT is omitted, file $OUTPUT is used.

VF

Specifies a valid vocabulary file. The descriptor of this file is system generated, causing the default value of this parameter to vary at sites that specify their own vocabulary file.

Omission of the VF parameter causes the default vocabulary file of your system to be used. If your site is using the vocabulary file supplied by IM/DM, DM$VOC: is the default for the VF parameter.

Remarks For more information, see the DM Utilities Reference Manual for DM on CDC NOS/VE.

DMCCF
IM/DM Command

Purpose Executes the DM Communications Control Facility to tune the communication software.

Format

DMCCF
ACTION=dm_question_mark or *keyword*
CCF=dm_file_descriptor
INPUT=dm_file_descriptor
OUTPUT=dm_file_descriptor
VF=dm_file_descriptor
STATUS=status variable

Parameters *ACTION* or *A*

Action to be performed on the communication control file (CCF).

CREATE

Creates a new CCF.

UPDATE

Updates an existing CCF.

READ

Displays an existing CCF.

If ACTION is omitted, READ is used.

CCF

Communications control file.

If CCF is omitted, file DMCCF_CCF is used.

INPUT

Input file containing any required parameters that are missing.

If INPUT is omitted, file INPUT is used.

OUTPUT

Output file on which all messages are written.

If OUTPUT is omitted, file $OUTPUT is used.

VF

Specifies a valid vocabulary file. The descriptor of this file is system generated, causing the default value of this parameter to vary at sites that specify their own vocabulary file.

Omission of the VF parameter causes the default vocabulary file of your system to be used. If your site is using the vocabulary file supplied by IM/DM, DM$VOC: is the default for the VF parameter.

Remarks For more information, see the DM System Administrator's Reference Manual for DM on CDC NOS/VE.

DMCPC
IM/DM Command

Purpose Begins a DM COBOL Precompiler (DMCPC) session which is used to compile special DM statements in the source code of COBOL program and produces calls to the DM user program interface (UPI).

Format

DMCPC
SOURCE=dm_file_descriptor
DECKS=list of range of name or *keyword*
PCPUT=dm_file_descriptor
OBJECT=dm_file_descriptor

ERROR=dm_file_descriptor
UID=dm_name
UPW=dm_name
PROGRAM=name
STMT=keyword
LINELC=integer
RECORDTYPE=keyword
RECORDLC=integer
TRACE=keyword
STATS=keyword
INPUT=dm_file_descriptor
VF=dm_file_descriptor
COBOL=keyword
COBOL_PARAMETERS=string
STATUS=status variable

Parameters

SOURCE or *S*

Sequential file or source library containing the source code. The precompiler expands the special *DM commands in the source code.

The source is assumed to be variable-length records unless specified otherwise by the RECORDTYPE and RECORDLC parameters.

If the file extension is omitted, _DMC is assumed. If the file source_DMC exists, it is used; otherwise, the source file is used. The precompiler default is a COBOL format source file.

IM/DM prompts you for this value if it is not specified.

DECKS or *D*

If the source file is a source library, specifying DECKS=ALL indicates that all the files in the source library are to be precompiled. If the source file is not a source library, the DECKS parameter performs no action. By default, if you specify a source library as the source file, you select the files to be precompiled.

PCPUT

Specifies the sequential file where the precompiled source code is written. The PCPUT file defaults to variable length records. Use the RECORDTYPE parameter to specify fixed records for the PCPUT file and the source file. Use the RECORDLC parameter to specify the length in characters for these files. If you specify the PCPUT

parameter without a file descriptor, the source code is written to the file source_COB (where source is a variable name). If you specify NO on the COBOL parameter, the file source_COB is created and saved; otherwise, by default, it is deleted after the COBOL compilation.

OBJECT or *OBJ*

The OBJECT parameter specifies the object file. Omission of the OBJECT parameter causes program_OBJ to be used.

ERROR

The ERROR parameter specifies the sequential file where error and warning messages are listed. Omission of the ERROR parameter causes OUTPUT to be used.

UID

The UID parameter specifies the user identification code. UID can be an integer or an identifier that contains letters and digits. Only the first 8 characters are used as the UID.

The UID parameter is required. DM prompts you for the UID parameter if you do not specify it on the command.

UPW

The UPW parameter specifies the user password. UPW can be an integer or an identifier that contains letters and digits. Only the first 8 characters are used as the UPW.

The UPW parameter is required. DM prompts you for the UPW parameter if you do not specify it on the command.

PROGRAM

Specifies an identifier that represents the name of the program being compiled. A program name entered on a *DM.PROGRAM statement within the GET file overrides this parameter.

STMT

Reserved.

LINELC

The LINELC parameter specifies the length in characters of the source lines of the language. LINELC is an integer value from 65 to 100. This overrides the global LINELC in the PCD.

Omission of the LINELC parameter causes 72 to be used.

RECORDTYPE

Specifies the record types for the source and PCPUT files. You can specify the keywords FIX and VAR. FIX indicates that the source and PCPUT files have fixed length records. VAR indicates that these files have variable length records. If RECORDTYPE is set to FIX, then the length in characters of the record is determined by the RECORDLC parameter.

Omission of the RECORDTYPE parameter causes VAR to be used.

RECORDLC

Specifies the length in characters of the records in the source and PCPUT files when RECORDTYPE is set to fixed. RECORDLC is an integer value from 1 to 200.

Omission of the RECORDLC parameter causes 200 to be used.

TRACE

The TRACE parameter specifies the action of the trace facility during program compilation. Options are:

NONE

Indicates that no tracing is performed.

REPLACE

Indicates that the precompiler should create program trace records for all the database models referenced. If a PTR for this program for a referenced model already exists, its views are replaced with the new views referenced by the current precompilation. You should use TRACE=REPLACE when the program is moved into production and when initial program trace records are created.

UPDATE

Indicates that the view reference lists in the program trace records are updated with any new views referenced during the current compile. If a new model is referenced, a new program trace record is created. You should use TRACE=UPDATE when new models and new views have been referenced within a module of the program. Only the module that contains the new references needs to be precompiled.

Omission of the TRACE parameter causes NONE to be used.

STATS

The STATS parameter indicates if the statistics for the precompile should be printed. STATS=NO indicates that no statistics are printed. STATS=YES indicates that the following statistics are printed:

ELAPSED SECONDS:
ELAPSED CPU SECONDS:
SOURCE LINES READ:
ROUTINES PROCESSED:
*DM STATEMENTS PROCESSED:

Omission of the STATS parameter causes NO to be used.

INPUT

Specifies the terminal or a sequential file of variable length records. It contains answers to the prompts produced by the program.

Omission of the INPUT parameter causes COMMAND to be used.

VF

Specifies a valid vocabulary file. The descriptor of this file is system generated, causing the default value of this parameter to vary at sites that specify their own vocabulary file.

Omission of the VF parameter causes the default vocabulary file of your system to be used. If your site is using the vocabulary file supplied by IM/DM, DM$VOC: is the default for the VF parameter.

COBOL or *COB*

Specifies whether or not to perform the COBOL compilation. You can specify the keywords YES or NO. YES means that a COBOL compilation is performed. NO means that a COBOL compilation is not performed.

If you do not specify the PCPUT parameter, the file is saved in the file source_COB (where source is a variable name). By default, if no errors were detected during the precompilation, a COBOL compilation is performed.

COBOL_PARAMETERS or *CP*

Specifies optional NOS/VE COBOL parameters. You can specify any of the NOS/VE COBOL parameters. See the COBOL Usage manual for more information.

Omission of the COBOL_PARAMETERS parameter causes only the COBOL parameters specified on the DMCPC command to be used.

Remarks

- The precompiler can update the definition database to identify which views are used by the program. This trace information can be used to locate all the programs that reference a particular view. Only authorized users can use the precompiler and unauthorized references to private models are not allowed.

- For more information, see the IM/DM Application Programming manual.

DMDBA
IM/DM Command

Purpose Executes the DM Database Administration module to enter and modify the definition of a database.

Format

DMDBA
UID=dm_name
UPW=dm_name
AIDS=keyword
DB=name
ACTION=keyword
MODE=keyword
TALK=integer
TERMINAL=dm_terminal

JOURNAL=keyword
APPLY_IF_OK=keyword
CANCEL_CHANGES=keyword
MENU=keyword
RECORDTYPE=keyword
RECORDLC=integer
LINELC=integer
GET=dm_file_descriptor
INPUT=dm_file_descriptor
OUTPUT=dm_file_descriptor
PRINT=dm_file_descriptor
VOCABULARY_FILE=dm_file_descriptor
STATUS=status variable

Parameters

UID

User identification code (only first 8 characters are used).

IM/DM prompts you for this value if it is not specified.

UPW

User password (only the first 8 characters are used).

IM/DM prompts you for this value if it is not specified.

AIDS

Indicates whether you are given the option of reviewing the user aids.

YES

Aids option is given.

NO

Aids option is not given.

Omission of this parameter when running a command procedure in statement mode causes YES to be used. If you are running the command procedure in dialog or screen mode, the job terminates. Omission of this parameter when running a job interactively causes DMDBA to prompt you for the AIDS value.

DB

Name of the definition database to be processed.

IM/DM prompts you for this value if it is not specified.

ACTION or *A*

Action to be performed on the database.

CREATE

New database created.

UPDATE

Existing database updated.

APPLY

Definition applied.

ADMIN

Administrative tasks performed.

This parameter is required except when running a job in statement mode, for which UPDATE is assumed.

MODE

Input mode used.

STATEMENTS

Data entered using a file of statements.

DIALOG

Data entered in response to prompts.

SCREEN

Data entered by filling in information on screens.

IM/DM prompts you for this value if it is not specified.

TALK

Level of message detail.

0

Brief messages.

1

Detailed messages.

Omission of this parameter when running a command procedure in statement mode causes 1 to be used. If you are running the command procedure in dialog or screen

mode, the job terminates. Omission of the TALK parameter when running a job interactively causes DMDBA to prompt you for the TALK value.

TERMINAL

Type of terminal interface.

CVTI

Character Virtual Terminal Interface.

*

Site default.

If TERMINAL is omitted, * is used.

JOURNAL

Indicates whether journaling is to be performed.

YES

Journaling is performed.

NO

Journaling is not performed.

If JOURNAL is omitted, YES is used.

APPLY_IF_OK

Indicates whether the database definition is applied.

YES

Apply is performed.

NO

Apply is not performed.

If APPLY_IF_OK is omitted, NO is used.

CANCEL_CHANGES

Indicates whether erroneous change orders are cancelled.

YES

Erroneous change orders are cancelled.

NO

Change orders are not cancelled.

If CANCEL_CHANGES is omitted, NO is used.

MENU

Carriage control before printing of the menu.

PAGE

Page feed before the menu is printed.

MENU

Clear screen before the menu is displayed.

SCROLL

Line feed before the menu is printed.

If MENU is omitted, SCROLL is used.

RECORDTYPE

Type of records in the GET file.

VAR

Variable-length records.

FIX

Fixed-length records.

If RECORDTYPE is omitted, VAR is used.

RECORDLC

Record length in characters for the GET file (integer).

If RECORDLC is omitted, 255 is used.

LINELC

Maximum length in characters of a line in the GET file (integer).

If LINELC is omitted, 250 is used.

GET

Specifies the file of statements defining the database to be used in the statement mode. DMDBA prompts you for this parameter if you do not specify it on the command. This parameter is required when ACTION=STATEMENTS is specified.

INPUT

File containing responses to program prompts.

If INPUT is omitted, file INPUT is used.

OUTPUT

File to which all error messages, prompts, and program output is written.

If OUTPUT is omitted, the standard output file, $OUTPUT, is used.

PRINT

Specifies a file that output is sent to if screens are printed. Omission causes the file DBA_PRINT_FILE to be used.

VOCABULARY_FILE or *VF*

Specifies the valid vocabulary file. The descriptor of this file is system generated, causing the default value of this parameter to vary at sites that specify their own vocabulary files.

Omission of the VF parameter causes the default vocabulary file of your system to be used. If your site is using the vocabulary file supplied by IM/DM, DM$VOC: is the default for the VF parameter.

Remarks For more information, see the DM Utilities Reference Manual for DM on CDC NOS/VE.

DMDDBD
IM/DM Command

Purpose Executes the DM Definition Database Dump module for a database administrator.

Format

DMDDBD
UID=dm_name
UPW=dm_name
DB=name
ACTION=keyword
INPUT=dm_file_descriptor
OUTPUT=dm_file_descriptor
VF=dm_file_descriptor
STATUS=status variable

Parameters *UID*

Specifies the user identification code. UID can contain eight letters and digits. This parameter is required. IM/DM prompts you for this value if it is not specified.

UPW

Specifies the user password. UPW can contain eight letters or digits. This parameter is required. IM/DM prompts you for this value if it is not specified.

DB

Name of the definition database to be processed.

IM/DM prompts you for this value if it is not specified.

ACTION

Definition database records to display.

SELECT

Records selected by key.

ALL

All records.

ALLKEYS

Keys of each record.

Omission of the ACTION parameter causes SELECT to be used.

INPUT

File containing responses to program prompts.

If INPUT is omitted, file INPUT is used.

OUTPUT

File to which all error messages, prompts, and program output are written.

If OUTPUT is omitted, the standard output file, $OUTPUT, is used.

VF

Specifies a valid vocabulary file. The descriptor of this file is system generated, causing the default value of this parameter to vary at sites that specify their own vocabulary file.

Omission of the VF parameter causes the default vocabulary file of your system to be used. If your site is using the vocabulary file supplied by IM/DM, DM$VOC: is the default for the VF parameter.

Remarks

- For more information, see the IM/DM Data Administration manual.

DMDDBE
IM/DM Command

Purpose

Executes the DM Definition Database Extract Utility to generate a DDL statement file for input to DMDBA from a definition database.

Format

DMDDBE
UID=dm_name
UPW=dm_name
DB=name
ACTION=keyword
OBJECT_TYPE=dm_name
OBJECT_NAME=dm_name
RECORDTYPE=keyword
RECORDLC=integer
PUT=dm_file_descriptor
INPUT=dm_file_descriptor
OUTPUT=dm_file_descriptor
VF=dm_file_descriptor
STATUS=status variable

Parameters

UID

User identification code (only first 8 characters are used).

IM/DM prompts you for this value if it is not specified.

UPW

User password (only the first 8 characters are used).

IM/DM prompts you for this value if it is not specified.

DB

Name of the definition database to be processed.

IM/DM prompts you for this value if it is not specified.

ACTION or *A*

Action the utility is to perform.

SELECT

Extracts more than one object, but not the entire definition database.

ALL

Extracts all objects in the database.

ONE

Extracts one object type from the database.

If ACTION is omitted, SELECT is used.

OBJECT_TYPE

Used when ACTION=ONE to specify the object type to be extracted.

OBJECT_NAME

Used when ACTION=ONE to specify the name of the object to be extracted.

Object_Type	Object_Name
ADM_ALL	None required.
SDM_ALL	None required.
UDM_ALL	None required.
FILES_ALL	None required.
MODEL_ALL	Model name Stem (all models)
RECORD_ALL	Record name Stem (all record types)
RECORD	Record name Stem (all record types)

WORD_LIST	Word list name Stem (all word lists)
CODE_LIST	Code list name Stem (all code lists)
ELEMENT_LIST	ADM.record name.element list ADM.record name.stem (all ADM and UDM element lists) ADM.record name. (all ADM and UDM element lists) ADM.stem (all ADM and UDM element lists) ADM. (all ADM and UDM element lists) Model name.view name.element list Model name.view name.stem (all ADM and UDM element lists) Model name.view name. (all ADM and UDM element lists) Model name.stem (all ADM and UDM element lists) Model name. (all ADM and UDM element lists) lists)
VIEW	Model name.view name Model name.stem (all models, all view types) Model name. (all models, all view types) Stem.view name (all models, all view types) Stem (all models, all view types) .view name (all models, all view types)
INDEX	Record name.element name Record name.stem (all indexes) Record name. (all indexes) Stem (all indexes)
ASSERT	Assert number:assert number Assert number * (all assertions)

RECORDTYPE

Record type for the file specified on the PUT parameter.

FIX

Fixed-length records.

VAR

Variable-length records.

If RECORDTYPE is omitted, VAR is used.

RECORDLC

Maximum record length in the PUT file.

If RECORDLC is omitted, 80 is used.

PUT

File containing the DDL statements created by DMDDBE.

If PUT is omitted, file DMDDBE_PUT is used. Specify a file name for this parameter.

INPUT

Input file containing any required parameters that are missing.

If INPUT is omitted, file INPUT is used.

OUTPUT

Output file on which all messages are written.

If OUTPUT is omitted, file $OUTPUT is used.

VF

Specifies a valid vocabulary file. The descriptor of this file is system generated, causing the default value of this parameter to vary at sites that specify their own vocabulary file.

Omission of the VF parameter causes the default vocabulary file of your system to be used. If your site is using the vocabulary file supplied by IM/DM, DM$VOC: is the default for the VF parameter.

Remarks For more information, see the DM Utilities Reference Manual for DM on CDC NOS/VE.

DMDDBR
IM/DM Command

Purpose

The DM Definition Database Report Utility (DMDDBR) generates reports that use the definition database as the source of information. The reports show the structure of the database and how parts of the database are related. The reports are listed on menus for easy selection. When the reports have been created, you can display them on the terminal or write them to an output file.

Format

DMDDBR
UID=dm_name
UPW=dm_name
INPUT=dm_file_descriptor
OUTPUT=dm_file_descriptor
STATUS=status variable

Parameters

UID

Specifies the user identification code. It can contain 8 letters and digits. This parameter is required.

UPW

Specifies the user password. It can contain 8 letters and digits. This parameter is required.

INPUT

Reserved.

OUTPUT

Reserved.

Remarks

- For more information, see the DM Utilities Reference Manual for DM on CDC NOS/VE.
- Entering DMDDBR has the same effect as entering the following command:

```
/dmrw dbprompt=no, aids=no, mtr=no, ..
../proc=dm$ddbr:mm_prc
```

DMDRL
IM/DM Command

Purpose Executes the DM Dump and Reload module for a database administrator.

Format

DMDRL
UID=dm_name
UPW=dm_name
DB=name
MODEL=name
ACTION=keyword
DUMPFILE=dm_file_descriptor
DATAFILE=dm_file_descriptor
FORMAT=dm_key or *keyword*
TLIB=dm_file_descriptor
STATS=keyword
TRACE=keyword
DELETE=keyword
RECDELIM=dm_character
ELEMDELIM=dm_character
SUBDELIM=dm_character
DELCODE=dm_character
STARTOCC=dm_integer
ENDOCC=dm_integer
VIEW=name
KEY=name
VERSION=integer
METHOD=keyword
CHECKREF=keyword
VALIDATE=keyword
JOURNAL=keyword
RECOVERY=keyword
TRANSLIMIT=integer
ERRORLIMIT=integer
ERRORACTION=keyword
REJECT=dm_file_descriptor
REJECTFORMAT=keyword
INPUT=dm_file_descriptor
OUTPUT=dm_file_descriptor
VF=dm_file_descriptor
STATUS=status variable

Parameters *UID*

Specifies the user identification code. UID contains eight letters or digits. IM/DM prompts you for this value if it is not specified.

UPW

Specifies the user password. UPW contains eight letters or digits. IM/DM prompts you for this value if it is not specified.

DB

Name of the database to be processed.

IM/DM prompts you for this value if it is not specified.

MODEL

Name of an FQM user data model that dumps or reloads the records.

IM/DM prompts you for this value if it is not specified.

ACTION or *A*

Action performed.

DUMP

Write database records to a data file.

PARTIAL_DUMP (PARTIAL)

Write selected database records to a data file.

RELOAD

Read records from a data file and place the records in the database.

SALVAGE

Write records to a data file from a database that has been damaged.

UPDATE

Update the contents of the database.

If ACTION is omitted, DUMP is used.

DUMPFILE

Name of the data file. IM/DM prompts you for this value if this parameter or the datafile parameter is not specified.

DATAFILE

Specifies the name of the data file. The format of this file is specified by the FORMAT parameter. Omission of the DATAFILE parameter means the DUMPFILE parameter must be specified.

FORMAT or *FORM*

Format of the data file.

DUMP

Binary file format.

FREE

Designated delimiter characters format.

STREAM

Sequential file format.

TEMPLATE

Specifies the layout of the file. Format is FORMAT=TEMPLATE (template_name), where template_name is the name of the template in the template library.

This parameter is required.

TLIB

This parameter specifies the name of the template library. A template library is a sequential file that contains templates. The templates define the format of data files and are used when the FORMAT parameter is set to TEMPLATE (template_name) where template_name is the name of the template.

STATS

Indicates whether run time statistics are displayed.

YES

Displays statistics.

NO

Does not display statistics.

If STATS is omitted, NO is used.

TRACE

This parameter specifies the explicitness of the trace messages displayed. You can choose from the following options:

FULL

Complete trace messages are displayed.

BRIEF

Brief trace messages are displayed.

NONE

No trace messages are displayed.

Omission causes BRIEF to be used.

DELETE

Indicates whether dumped records are deleted from the database.

YES

Deletes records.

NO

Does not delete records.

If DELETE is omitted when ACTION=DUMP, NO is used.

RECDELIM

This parameter specifies the character to delimit record occurrences in the data file when the format of the file is free (FORMAT=FREE). The DATAFILE parameter specifies the file containing the data file. Omission of the RECDELIM parameter when the FORMAT parameter is set to FREE causes the character @ to be read.

ELEMDELIM

Specifies the character to delimit record occurrences in the data file when the format of the file is free (FORMAT=FREE). The DATAFILE parameter specifies

the file containing the data file. Omission of this parameter when the FORMAT parameter is set to FREE causes the character # to be read.

SUBDELIM

This parameter specifies the character to delimit subelement values in the data file when the format of the file is free (FORMAT=FREE). The DATAFILE parameter specifies the file containing the data file. Omission of this parameter when the FORMAT parameter is set to FREE causes the character ; to be used.

DELCODE

This parameter specifies the character to indicate element values to delete in the data file when the format of the file is free (FORMAT=FREE). The DATAFILE parameter specifies the file containing the data file. Omission of the DELCODE parameter when the FORMAT parameter is set to FREE causes the character \ to be used.

STARTOCC

Specifies on which record occurrence in the data file processing is to start. Must be an integer. You can specify this parameter when reloading or updating a database (ACTION=RELOAD) or (ACTION=UPDATE). Omission of this parameter when ACTION is set to RELOAD or UPDATE causes processing to begin with the first record occurrence in the data file.

ENDOCC

Specifies on which record occurrence in the data file processing is to stop. ENDOCC must be an integer.

You can specify the ENDOCC parameter when reloading or updating a database (ACTION=RELOAD or ACTION=UPDATE). Omission of this parameter when ACTION is set to RELOAD or UPDATE causes processing to stop on the last record occurrence in the data file.

VIEW

Name of the view used to read records from the database.

This parameter is required when ACTION=DUMP or ACTION=SALVAGE.

KEY

When ACTION=DUMP, the KEY parameter must specify one of the following commands or methods: PUT, REPLACE, DELETE, or UPDATE. This parameter specifies the name of a key element to use for the PUT, REPLACE, DELETE, and UPDATE commands. A key element is the element that uniquely identifies a record occurrence.

VERSION

Version number of the default database (0 through 99).

If VERSION is omitted, 0 is used.

METHOD or *COMMAND*

This parameter is included for compatability with previous releases of IM/DM. The COMMAND parameter replaces the METHOD parameter.

This parameter specifies the command to use when the input record of the datafile does not specify a command. You can specify:

ADD

Adds a new record occurrence to the database.

PUT

Puts a new record occurrence in the database whether or not a record occurrence with the same unique key already exists. The unique key must be identified by the KEY parameter of the DMHVL command or identified in the input record.

REPLACE

Replaces an existing record occurrence with a record occurrence listed in a data file.

DELETE

Deletes an existing record occurrence.

UPDATE

Updates an existing record occurrence.

This parameter is required for data files in the DUMP format because the DUMP format does not allow command values. You can specify this parameter when

reloading or updating a database. Omission of this parameter when the ACTION parameter is set to RELOAD or UPDATE causes PUT to be used.

CHECKREF

Indicates whether referential integrity constraints are checked when data is loaded.

YES

Constraints are checked.

NO

Constraints are not checked.

If CHECKREF is omitted, YES is used.

VALIDATE

Indicates whether the system should perform records and element validation checks when data is loaded.

YES

Validations are performed.

NO

Validations are not performed.

If VALIDATE is omitted, YES is used.

JOURNAL

Indicates whether journaling is performed when data is loaded.

YES

Journaling is performed.

NO

Journaling is not performed.

If JOURNAL is omitted, YES is used.

RECOVERY

Indicates whether automatic recovery is enabled in the case of system failure.

YES

Recovery is enabled.

NO

Recovery is not enabled.

If RECOVERY is omitted, YES is used.

TRANSLIMIT

This parameter specifies the transaction size used by DMDRL. The value can be an integer ranging from 1 to 20,000. DMDRL dumps or reloads a database using numerous transactions. When the transaction limit is reached, a FINISH is performed to end the transaction. A START is then performed to begin a new transaction. Omission of the TRANSLIMIT parameter causes 20,000 to be used.

ERRORLIMIT or *EL*

Maximum number of errors that occur before the program terminates. The value can be an integer ranging from 1 to 1,000,000.

You can specify the ERRORLIMIT parameter when reloading or updating a database (ACTION=RELOAD or ACTION=UPDATE). Omission of the ERRORLIMIT parameter when ACTION=RELOAD or ACTION=UPDATE causes 1000 to be used.

ERRORACTION

Indicates whether database changes are kept if the error limit is reached.

ABORT

All changes are rolled back.

FINISH

All changes made are retained.

ERRORACTION is used when ACTION=RELOAD or ACTION=UPDATE. If you omit ERRORACTION when ACTION=RELOAD or ACTION=UPDATE, ABORT is used.

REJECT

Specifies a file to which records containing errors are written. The records are written in STREAM format. The records can be edited and then reloaded. You can specify

this parameter when reloading or updating a database (ACTION=RELOAD or ACTION=UPDATE). Omission of this parameter causes DMDRL_REJ to be used.

REJECTFORMAT

Specifies the format of the reject file, which is specified by the REJECT parameter. Formats are:

STREAM

A sequential file where each record of the file provides values for data elements. Long values can span several records of a file.

INPUT

The rejected records are written as they appear in the data file. The format of the data file is specified by the format parameter.

Use this parameter only when you are reloading or updating a database (ACTION=RELOAD) or (ACTION=UPDATE). Omission of this parameter when ACTION is set to RELOAD or UPDATE causes STREAM to be used. If the data file references multiple view types (MULTIVIEW=YES), the reject format should be INPUT.

INPUT

Specifies the input file. It should contain required parameters not specified on the DMDRL command. If INPUT is omitted, file INPUT is used.

OUTPUT

File to which all error messages, prompts, and program output are written.

If OUTPUT is omitted, the standard output file, $OUTPUT, is used.

VF

Specifies a valid vocabulary file. The descriptor of this file is system generated, causing the default value of this parameter to vary at sites that specify their own vocabulary file.

Omission of the VF parameter causes the default vocabulary file of your system to be used. If your site is using the vocabulary file supplied by IM/DM, DM$VOC: is the default for the VF parameter.

Remarks For more information, see the DM Utilities Reference Manual for DM on CDC NOS/VE.

DMEMS
IM/DM Command

Purpose The DMEMS command calls the DM Error Message Summary (DMEMS) utility which reads through the vocabulary file and prints out the error messages.

Format

DMEMS
MSG=range of dm_integer
LEVEL=keyword
I=keyword
HEADER=keyword
EXACT=keyword
OUTPUT=dm_file_descriptor
VF=dm_file_descriptor
STATUS=status variable

Parameters *MSG*

Specifies the range of messages to print. MSG can be any of the following:

m

Where m is an integer value from 1 to 90,000.

m:n

Where m and n are integer values that specify a range of messages from 1 to 90,000. M must be less than n.

m:*

Where m is an integer value from 1 to 90,000. An asterisk indicates the last message the file contains.

*

An asterisk by itself means that all the messages in the file are to be printed, subject to the settings of the other parameters.

Omission of the MSG parameter causes an asterisk to be used and all the messages in the file are printed.

LEVEL

Specifies the level of messages to be printed. If you specify LEVEL=ANY, all levels of the messages are printed. If you do not specify the LEVEL parameter, only messages at level 3 and above are printed.

I

Specifies whether or not type I (INFORMATIVE) messages are printed. You can specify the keywords YES or NO. YES indicates that the messages of type I, along with the UE (USER ERROR) and SE (SYSTEM ERROR) messages above 5000, are printed. NO indicates that only type UE and SE messages above 5000 are printed.

Omission of the I parameter causes NO to be used.

HEADER

Specifies whether or not the DMEMS heading, page headers, and numbers are printed. You can specify the keywords YES or NO. YES indicates that the heading, page headers, and numbers are printed. NO means they are not printed.

Omission of the HEADER parameter causes YES to be used.

EXACT

Specifies how the error messages are printed. You can specify the keywords YES or NO. For EXACT=NO, $S, $P, and $Q are treated as $B. For EXACT=YES, $S causes lines to be skipped. Also, column 2 of the first output line for the message contains the level number, column 3 contains Q for $Q, and column 4 contains M, if MORE=YES is specified on the message. Messages of type UE or SE have the appropriate prefix attached to the output line.

Omission of the EXACT parameter causes NO to be used.

OUTPUT

Specifies an alternate output file to which the listing is sent. Omission of the OUTPUT parameter causes $OUTPUT to be used.

VF

Specifies a valid vocabulary file. The descriptor of this file is system generated, causing the default value of this parameter to vary at sites that specify their own vocabulary file.

Omission of the VF parameter causes the default vocabulary file of your system to be used. If your site is using the vocabulary file supplied by IM/DM, DM$VOC: is the default for the VF parameter.

Remarks

- All printing is done between columns 6 and 85 of the output file to allow for a left margin. However, if HEADER=NO, printing begins in column 1. At most, 55 lines are put on a page and only 50 lines of a message can be printed. All $G and $C edits in a message are ignored and parameter entries are replaced with asterisks (*) up to the length specified in the picture. The first line of message output is:

  type message number ID=id

  This is followed by the levels of the message separated by a blank line. The actual format of the message text is determined by the setting of the EXACT parameter.

- For more information, see the IM/DM Application Programming manual.

DMFORM
IM/DM Command

Purpose Begins DM Form View Development utility session to define and maintain form views.

Format

DMFORM
UID=dm_name
UPW=dm_name
DB=name
AIDS=keyword
TERMINAL=dm_terminal
TALK=keyword

MRT=keyword
PROC=dm_file_descriptor
INPUT=dm_file_descriptor
OUTPUT=dm_file_descriptor
VF=dm_file_descriptor
STATUS=status variable

Parameters *UID*

User identification code. It can be an integer or an identifier that contains eight letters and digits.

UPW

User password. It can be an integer or an identifier that contains eight letters and digits.

DB

Name of the default definition database to be processed.

AIDS

Indicates whether you are given the option of reviewing the user aids.

YES

Aids option is given.

NO

Aids option is not given.

If AIDS is omitted, YES is used.

TERMINAL

Type of terminal interface.

CVTI

Character Virtual Terminal Interface (CVTI). CVTI tells DM the terminal you are using and enables the DM screen features to work correctly on your terminal. CVTI can be specified for any terminal defined in the system by the TERMINAL_DEFINITION_UTILITY.

*

Site default terminal.

If TERMINAL is omitted, * is used.

TALK

Initial level of message detail.

EXPERT

Brief messages.

STANDARD

Normal messages.

NOVICE

Detailed messages.

If TALK is omitted, STANDARD is used.

MRT

Indicates whether the module revision tag is printed at the start of DMFORM execution.

YES

The module revision tag is printed.

NO

The module revision tag is not printed.

If MRT is omitted, YES is used.

PROC

Name of an optional file containing a DM command procedure to be executed.

INPUT

File containing responses to DMFORM prompts.

If INPUT is omitted, file INPUT is used.

OUTPUT

File to which all error messages, prompts, and program output is written.

If OUTPUT is omitted, the standard output file, $OUTPUT, is used.

VF

Specifies a valid vocabulary file. The descriptor of this file is system generated, causing the default value of this parameter to vary at sites that specify their own vocabulary file.

Omission of the VF parameter causes the default vocabulary file of your system to be used. If your site is using the vocabulary file supplied by IM/DM, DM$VOC: is the default for the VF parameter.

Remarks For more information, see the DM Utilities Reference Manual for DM on CDC NOS/VE.

DMFPC
IM/DM Command

Purpose The DMFPC command begins a DM FORTRAN precompiler (DMFPC) session that compiles special DM statements in the source code of a FORTRAN program and produce calls to the DM user program interface (UPI).

Format

DMFPC
SOURCE=dm_file_descriptor
DECKS=list of range of name or *keyword*
PCPUT=dm_file_descriptor
OBJECT=dm_file_descriptor
ERROR=dm_file_descriptor
UID=dm_name
UPW=dm_name
PROGRAM=name
STMT=keyword
LINELC=integer
RECORDTYPE=keyword
RECORDLC=integer
TRACE=keyword
STATS=keyword
INPUT=dm_file_descriptor
VF=dm_file_descriptor
FORTRAN=keyword
FORTRAN_PARAMETERS=string
STATUS=status variable

Parameters

SOURCE or *S*

Sequential file or source library containing the source code. The precompiler expands the special *DM commands in the source code.

The source is assumed to be variable-length records unless specified otherwise by the RECORDTYPE and RECORDLC parameters.

If the file extension is omitted, _DMF is assumed. If the file source_DMF exists, it is used; otherwise, the source file is used.

IM/DM prompts you for this value if it is not specified.

DECKS or *D*

If the source file is a source library, specifying DECKS=ALL indicates that all the files in the source library are to be precompiled. If the source file is not a source library, the DECKS parameter performs no action. By default, if you specify a source library as the source file, you select the files to be precompiled.

PCPUT

Specifies the sequential file where the precompiled source code is written. The PCPUT file defaults to variable length records. Use the RECORDTYPE parameter to specify fixed records for the PCPUT file and the source file. Use the RECORDLC parameter to specify the length in characters for these files. If the PCPUT parameter is entered without a file descriptor, the source code is written to the file source_FOR (where source is a variable name). If you specify NO on the FORTRAN parameter, the file source_FOR is created and saved; otherwise, by default, it is deleted after the FORTRAN compilation.

OBJECT or *OBJ*

Specifies the object file. Omission of the OBJECT parameter causes program_OBJ to be used.

ERROR

Specifies the sequential file where error and warning messages are listed. Omission of the ERROR parameter causes OUTPUT to be used.

UID

The UID parameter specifies the user identification code. UID can be an integer or an identifier that contains letters and digits. Only the first 8 characters are used as the UID.

The UID parameter is required. DM prompts you for the UID parameter if you do not specify it on the command.

UPW

Specifies the user password. UPW can be an integer or an identifier that contains letters and digits. Only the first 8 characters are used as the UPW.

The UPW parameter is required. DM prompts you for the UPW parameter if you do not specify it on the command.

PROGRAM

Specifies an identifier that represents the name of the program being compiled. A program name entered on a *DM.PROGRAM statement within the GET file overrides this parameter.

STMT

Reserved.

LINELC

Specifies the length in characters of the source lines of the language. LINELC is an integer value from 65 to 100. This overrides the global LINELC in the PCD.

Omission of the LINELC parameter causes 72 to be used.

RECORDTYPE

Specifies the record types for the source and PCPUT files. You can specify the keywords FIX or VAR. FIX indicates that the source and PCPUT files have fixed length records. VAR indicates that these files have variable length records. If RECORDTYPE is set to FIX, then the length in characters of the record is determined by the RECORDLC parameter.

Omission of the RECORDTYPE parameter causes VAR to be used.

RECORDLC

Specifies the length in characters of the records in the source and PCPUT files when RECORDTYPE is set to fixed. RECORDLC is an integer value from 1 to 200.

Omission of the RECORDLC parameter causes 200 to be used.

TRACE

The TRACE parameter specifies the action of the trace facility during program compilation. Options are:

NONE

Indicates that no tracing is performed.

REPLACE

Indicates that the view reference lists in the program trace records are replaced with the views referenced by the current compilation. If there are program trace records for public models no longer referenced, they are deleted.

UPDATE

Indicates that the view reference lists in the program trace records are updated with any new views referenced during the current compilation. If a new model is referenced, a new program trace record is created.

Omission of the TRACE parameter causes NONE to be used.

STATS

Indicates if the statistics for the precompilation should be printed. You can specify the keywords YES or NO. STATS=YES indicates that statistics are printed. NO indicates that no statistics are printed.

The statistics are:

ELAPSED SECONDS:
ELAPSED CPU SECONDS:
SOURCE LINES READ:
ROUTINES PROCESSED:
*DM STATEMENTS PROCESSED:

Omission of the STATS parameter causes NO to be used.

INPUT

Specifies the terminal or a sequential file of variable length records. It contains answers to the prompts produced by the program.

Omission of the INPUT parameter causes COMMAND to be used.

VF

Specifies a valid vocabulary file. The descriptor of this file is system generated, causing the default value of this parameter to vary at sites that specify their own vocabulary file.

Omission of the VF parameter causes the default vocabulary file of your system to be used. If your site is using the vocabulary file supplied by IM/DM, DM$VOC: is the default for the VF parameter.

FORTRAN or *FOR*

Specifies whether or not to perform the FORTRAN compilation. You can specify the keywords YES or NO. YES indicates that a FORTRAN compilation is performed. NO indicates that a FORTRAN compilation is not performed.

If you do not specify the PCPUT parameter, the PCPUT file is saved in the file source_FOR (where source is a variable name). By default, if no errors were detected during the precompilation, a FORTRAN compilation is performed.

FORTRAN_PARAMETERS or *FP*

Specifies optional NOS/VE FORTRAN parameters. You can specify any of the NOS/VE FORTRAN parameters. See the FORTRAN Language Definition manual for detailed information on the FORTRAN command parameters.

Remarks

- DMFPC can update the definition database to identify which views are used by the program. This trace information can be used to locate all the programs that reference a particular view. Only authorized users can use the precompiler and unauthorized references to private models are not allowed.

- For more information, see the IM/DM Application Programming manual.

DMFQM
IM/DM Command

Purpose Executes the DMFQM module to access the IM/DM Query Facility.

Format

DMFQM
UID=dm_name
UPW=dm_name
AIDS=keyword
DB=dm_database_model
INTENT=keyword
PROC=dm_file_descriptor
TALK=keyword
MRT=keyword
TERMINAL=dm_terminal_name
VERSION=integer
WSLC=integer
INPUT=dm_file_descriptor
OUTPUT=dm_file_descriptor
VF=dm_file_descriptor
STATUS=status variable

Parameters

UID

User identifier. If omitted, DMFQM prompts for the UID.

UPW

User password. If omitted, DMFQM prompts for the UPW.

AIDS

Indicates if DMFQM prompts for the option to review features and terminology. The default is YES.

DB

Default database name and user data model. The format of this parameter is database.model. If omitted, DMFQM prompts for the DB.

INTENT

Indicates how to use the database specified by the DB parameter. Keywords are:

UPDATE

Changes can be made to the database.

READ

Data can be searched and displayed only.

EXCLUSIVE

Changes can be made to the database and no one else may open the database.

USER

The intent is based on the user's privileges and the database privileges. If possible, UPDATE intent is used; otherwise, READ intent is used.

If INTENT is omitted, USER is used.

PROC

Name of the file containing the command procedure to be automatically executed at the start of the DMFQM session. If omitted, no command procedure is executed.

TALK

Level of expertise assumed by the DMFQM messages and user prompts. Keywords are:

EXPERT

Brief messages.

STANDARD

Descriptive messages (default).

NOVICE

Most descriptive messages.

MRT

Indicates whether DMFQM displays the Module Revision Tag when it begins the session. The default is YES.

TERMINAL

Type of terminal interface.

CVTI

Character Virtual Terminal Interface (CVTI). CVTI tells DM the terminal you are using and enables the DM screen features to work correctly on your terminal. CVTI can be specified for any terminal defined in the system by the TERMINAL_DEFINITION_UTILITY.

*

Site default terminal.

If TERMINAL is omitted, * is used.

VERSION

Version number of the occurrence database to be referenced. Version is an integer from 0 to 99. If omitted, 0 is used.

WSLC

The size in characters of the local work space. If omitted, DM uses the default size set by the System Administrator.

INPUT

Name of the sequential file that contains DMFQM commands to be used as input. If omitted, DMFQM reads its input from the terminal.

OUTPUT

Name of the sequential file to which DM writes messages and prompts. If omitted, DMFQM writes to the terminal.

VF

Name of the vocabulary file. If omitted, the site default vocabulary file is used.

Remarks

For more information, see the DM Fundamental Query and Manipulation manual.

Examples The following shows the beginning of a DMFQM session:

```
/DMFQM
DMFQM  V1  R12  851128 LIB(D381 S73  A0  )  830901
  user id> spc
  user pw> crud
  database> acme
  user model> fqma
The <ACME.FQMA> user model is open with UPDATE intent.

Do you want to review the user aids of this program
(YES or NO)? > n
DMFQM>
```

DMG
IM/DM Command

Purpose Executes the DM System Generation module to tune the system.

Format

DMG
ACTION=dm_question_mark or *keyword*
GET=dm_file_descriptor
PUT=dm_file_descriptor
RECORDTYPE=keyword
KERNELS=integer
USERS=integer
OUTPUT=dm_file_descriptor
STATUS=status variable

Parameters *ACTION*

Action performed:

ADB

Generates a skeleton authority database.

DECKS

Generates system routines for DMLIB.

FULL

Performs both the DECKS and ADB actions.

REGEN

Generates system routines and updates the authority database.

KMUWF

Generates the user work files for the kernels (not necessary for NOS/VE).

FQMUWF

Generates the user work files for the DMFQM module (not necessary for NOS/VE).

If ACTION is omitted, FULL is used.

GET

File containing the source statements of the system generation parameters to be set. It is used by the ADB, DECKS, FULL, and REGEN actions.

If GET is omitted, file GET is used.

PUT

File to which the generated routines are written by the DECKS, FULL, and REGEN actions.

If PUT is omitted, file PUT is used.

RECORDTYPE

Record type of the PUT file.

FIX

Fixed-length records of 72 characters.

VAR

Variable-length records.

If RECORDTYPE is omitted, VAR is used.

KERNELS

Number of kernels used in the system (1 through 9). Use this parameter only for the KMUWF and FQMUWF actions.

If KERNELS is omitted, 1 is used.

USERS

Maximum number of users allowed on a kernel (1 through 511). Use this parameter only for the KMUWF and FQMUWF actions.

OUTPUT

Output file on which all messages are written.

If OUTPUT is omitted, file $OUTPUT is used.

Remarks For more information, see the DM Utilities Reference Manual for DM on CDC NOS/VE.

DMHELP
IM/DM Command

Remarks Reserved for site personnel, Control Data, or future use.

DMHVL
IM/DM Command

Purpose The DM High Volume Loader, DMHVL, can quickly load or update a database. With DMHVL, record occurrences can be added, modified, or deleted without involving the kernel. A sequential data file or a dump file provides DMHVL with data and instructions for loading or updating the data. The database is accessed through an FQM model.

Format **DMHVL**

UID=dm_name
UPW=dm_name
DB=name
MODEL=name
ACTION=keyword
DATAFILE=dm_file_descriptor
FORMAT=dm_key or *keyword*
TLIB=dm_file_descriptor
COMMAND=keyword
STATS=keyword
TRACE=keyword
DELETE=keyword
RECDELIM=dm_character
ELEMDELIM=dm_character
SUBDELIM=dm_character
DELCODE=dm_character
STARTOCC=dm_integer
ENDOCC=dm_integer
VIEW=name
KEY=name

VERSION=integer
METHOD=keyword
CHECKREF=keyword
VALIDATE=keyword
JOURNAL=keyword
REJECT=dm_file_descriptor
REJECTFORMAT=keyword
INPUT=dm_file_descriptor
OUTPUT=dm_file_descriptor
VF=dm_file_descriptor
STATUS=status variable

Parameters

UID

Specifies the user identification code. UID contains eight letters or digits. IM/DM prompts you for this value if it is not specified.

UPW

Specifies the user password. UPW contains eight letters or digits. IM/DM prompts you for this value if it is not specified.

DB

Specifies the name of the database to load. This parameter is required.

MODEL

Specifies the name of a user data model that dumps and reloads the records. This parameter is required.

ACTION or *A*

Specifies how the data is loaded into the database.

UPDATE

Recomputes the virtual elements specified as SET_WHEN=ADD in the ADM for new records.

RELOAD

Does not recompute the virtual elements specified as SET_WHEN=ADD in the ADM for new records.

If ACTION is omitted, UPDATE is used.

DATAFILE

Specifies the name of the file that contains the data to be processed and loaded into the database. The format of this file is specified by the FORMAT parameter. This parameter is required.

FORMAT

Specifies the format of the data in the data file (the file specified on the DATAFILE parameter).

STREAM

A sequential file where each record of the file provides values for data elements.

DUMP

A file that contains non-character values and therefore cannot be edited with a conventional editor.

FREE

A file that uses designated delimiter characters to indicate the start of a new record, a new element value, a new subelement value, and element values to delete.

TEMPLATE

Defines the layout of the file. The template is stored in the template library which is a sequential file. The file can contain character data or binary values. To use this format, specify FORMAT=TEMPLATE (template_name) where template_name is the name of the template in the template library.

This parameter is required.

TLIB

Specifies the name of the template library. A template library is a sequential file containing HVL templates that are used when the data file uses the template format (FORMAT=TEMPLATE).

COMMAND

This parameter specifies the command to use when the input record of the datafile does not specify a command. This parameter is required when the data file uses the dump format because these files do not have a command value. You can specify:

ADD

Adds a new record occurrence to the database.

PUT

Puts a new record occurrence in the database whether or not a record occurrence with the same unique key already exists. The unique key must be identified by the KEY parameter of the DMHVL command or identified in the input record.

REPLACE

Replaces an existing record occurrence with a record occurrence listed in a data file.

DELETE

Deletes an existing record occurrence.

UPDATE

Updates an existing record occurrence.

Omission of the COMMAND parameter causes PUT to be used.

STATS

Indicates whether run time statistics are displayed.

YES

Displays statistics.

NO

Does not display statistics.

If STATS is omitted, NO is used.

TRACE

This parameter specifies the explicitness of the trace messages displayed. You can choose from the following options:

FULL

Complete trace messages are displayed.

BRIEF

Brief trace messages are displayed.

NONE

No trace messages are displayed.

Omission of the TRACE parameter causes BRIEF to be used.

DELETE

Reserved.

RECDELIM

Specifies the character that delimits record occurrences in the data file. This parameter is only valid when the data file uses the free format (FORMAT=FREE). Omission of this parameter when FORMAT=FREE causes @ to be used.

ELEMDELIM

Specifies the character that delimits element occurrences in the data file. This parameter is only valid when the data file uses the free format (FORMAT=FREE). Omission of this parameter when FORMAT=FREE causes # to be used.

SUBDELIM

This parameter specifies the character to delimit subelement values in the data file when the format of the file is free (FORMAT=FREE). Omission of this parameter when the FORMAT parameter is set to FREE causes the character ; to be used.

DELCODE

Specifies the character in the data file that indicates the element value is being deleted. This parameter is only valid for data files that use the free format (FORMAT=FREE). Omission of the DELCODE parameter when the FORMAT parameter is set to FREE causes the character \ to be used.

STARTOCC

Specifies the record occurrence in the data file on which the processing is to start. Omission causes processing to start with the first record occurrence in the data file (STARTOCC=1).

ENDOCC

Specifies on which record occurrence in the data file processing is to stop. ENDOCC must be an integer. Omission of this parameter causes the processing to continue through the end of the data file.

VIEW

Reserved.

KEY

Specifies the name of the default key element that uniquely identifies a record occurrence. This parameter is required when the data file uses the dump format (FORMAT=DUMP) and the PUT, REPLACE, DELETE, or UPDATE commands are performed. This parameter is optional when the STREAM, FREE, or TEMPLATE data file format is used. If this parameter is omitted, the unique key element must be specified in the data file.

VERSION

Version number of the default database (0 through 99).

If VERSION is omitted, 0 is used.

METHOD

Reserved.

CHECKREF

Specifies whether or not referential integrity constraints are checked when data is loaded. You can specify the keywords YES or NO.

If an index used in a referential integrity constraint was dropped, the constraint cannot be checked and CHECKREF=NO must be specified. DMDRL opens the database with EXCLUSIVE intent if you specify CHECKREF=NO.

YES

Referential integrity constraints are checked.

NO

Referential integrity constraints are not checked.

If CHECKREF is omitted, YES is used.

VALIDATE

Indicates whether the system should check if the data is valid.

YES

Validations are performed.

NO

Validations are not performed.

If VALIDATE is omitted, NO is used.

JOURNAL

Reserved.

REJECT

Specifies the name of the file on which DM puts records that contain errors. The records are written in the format specified by the REJECTFORMAT parameter. Omission of this parameter causes the file DMHVL_REJ to be used.

REJECTFORMAT

Specifies the format of the reject file, which is specified by the REJECT parameter. Formats are:

STREAM

Converts the rejected record to STREAM format.

INPUT

The rejected records are written as they appear in the data file.

Omission of this parameter causes STREAM to be used. If the data file references multiple view types (MULTIVIEW=YES), the reject format should be INPUT.

INPUT

Specifies the input file. It contains answers to user prompts. If INPUT is omitted, file INPUT is used.

OUTPUT

Specifies the output file. This file lists all messages produced by the program. Omission of the OUTPUT parameter causes $OUTPUT to be used.

VF

Specifies a valid vocabulary file. The descriptor of this file is system generated, causing the default value of this parameter to vary at sites that specify their own vocabulary file.

Omission of the VF parameter causes the default vocabulary file of your system to be used. If your site is using the vocabulary file supplied by IM/DM, DM$VOC: is the default for the VF parameter.

Remarks For more information, see the DM Utilities Reference Manual for DM on CDC NOS/VE.

DMJ
IM/DM Command

Purpose Executes the DM Journal Processor module to replay or backout changes made to a database.

Format

DMJ
UID=dm_name
UPW=dm_name
DB=name
ACTION=keyword
JOURNAL=list of dm_file_descriptor
VERSION=integer
DATE=dm_date
TIME=integer
SUMMARY=keyword
STATFILE=dm_file_descriptor
INPUT=dm_file_descriptor
OUTPUT=dm_file_descriptor
VF=dm_file_descriptor
STATUS=status variable

Parameters *UID*

User identification code (only the first 8 characters are used).

UPW

User password (only the first 8 characters are used).

DB

Database on which the action is performed.

This parameter is required for all actions except SYNCLIST.

ACTION

Action to be performed.

REPLAY

Reads journals and applies committed transactions in sequence.

BACKOUT

Reads journals backward, removing the committed changes in reverse sequence.

SYNCLIST

Lists the synchronization points in the journal files.

STATISTICS

Lists the information gathered in the journal files about database usage.

IM/DM prompts you for this value if it is not specified.

JOURNAL or *JOURNALS*

Journal files to be processed.

IM/DM prompts you for this value if it is not specified.

VERSION

Database version (0 through 99) used when ACTION=REPLACE or BACKOUT.

If VERSION is omitted, 0 is used.

DATE

Date in standard DM date format.

TIME

Time in the format HHMMSS.

This parameter must be specified when the DATE parameter is specified.

SUMMARY

Indicates whether a summary listing of elapsed time, CPU time, and images processed is given.

YES

Gives a summary.

NO

Does not give a summary.

If SUMMARY is omitted, NO is used.

STATFILE

File to which the statistics are written when ACTION=STATISTICS.

If STATFILE is omitted, file DMJ_STA is used.

INPUT

File from which any missing required parameters are read.

If INPUT is omitted, file INPUT is used.

OUTPUT

File to which all messages are written.

If OUTPUT is omitted, the standard output file, $OUTPUT, is used.

VF

Specifies a valid vocabulary file. The descriptor of this file is system generated, causing the default value of this parameter to vary at sites that specify their own vocabulary file.

Omission of the VF parameter causes the default vocabulary file of your system to be used. If your site is using the vocabulary file supplied by IM/DM, DM$VOC: is the default for the VF parameter.

Remarks For more information, see the DM Utilities Reference Manual for DM on CDC NOS/VE.

DMKMON
IM/DM Command

Purpose The DM Kernel Monitor Utility (DMKMON) monitors the activity of the kernel. Reports generated from within DMKMON show both the type and amount of kernel activity. The reports are updated periodically to provide the user with current information about the kernel. Report information can be displayed at the terminal or saved in a file and used to generate reports later. Anyone with a valid DM user ID and a valid user password can use this utility.

Format

DMKMON
UID=dm_name
UPW=dm_name
AIDS=keyword
REPORT=keyword
START=keyword
INTERVAL=integer
RUNTIME=integer
KERNEL=dm_kernel
TERMINAL=dm_terminal_name
ACTION=keyword
RECORDFILE=dm_file_descriptor
INPUT=dm_file_descriptor
OUTPUT=dm_file_descriptor
VF=dm_file_descriptor
STATUS=status variable

Parameters *UID*

Specifies the user identification code. UID must begin with a letter and can contain up to eight letters and digits. IM/DM prompts you for UID if it is not specified.

The UID parameter is required.

UPW

Specifies the user password. UPW must begin with a letter and can contain up to eight letters and digits. IM/DM prompts you for UPW if it is not specified.

The UPW parameter is required.

AIDS

Indicates whether you are given the option of reviewing the user aids available. Options are YES or NO.

Omission of the AIDS parameter causes YES to be used.

REPORT

Specifies the name of a monitor report. DMKMON can produce numerous reports; this parameter indicates which report to execute. The REPORT parameter is not applicable when ACTION=PLAYBACK. REPORT options are:

ACTIVITY
CACHE
DATABASE
FILES
LOCK_SUMMARY
LOCK_PROCESS
LOCK_USER
PROCESS
REQUEST_COUNTS
REQUEST_RATES
USER

Omission of this parameter causes ACTIVITY to be used.

START

Determines the starting point of report calculations. Report calculations can be based on the start of execution of the report or the start of execution of the kernel. Options are:

START=REPORT

Calculations are based on the start of the report's execution.

START=KERNEL

Calculations are based on the start of the kernel's execution.

Omission of this parameter causes START=KERNEL to be used.

INTERVAL

Reserved.

RUNTIME

Establishes the elapsed execution time in seconds. RUNTIME controls the elapsed execution time of DMKMON when DMKMON is run in non-interactive mode. If the parameter INPUT is set to a terminal, this parameter is ignored. Allowed values of elapsed time are 0 to 1000000.

Omission of the RUNTIME parameter causes DMKMON to execute for 3600 seconds.

KERNEL

Specifies the number of the monitored kernel. The number must be between 1 and 9 inclusive. If a kernel number is specified, the indicated kernel within the kernel set of the user's primary kernel is monitored.

Omission of the KERNEL parameter causes the user's primary kernel to be monitored.

TERMINAL

Specifies the type of terminal on which the DMKMON reports are displayed.

CVTI

Character Virtual Terminal Interface. CVTI informs DMKMON of the type of terminal you are using and enables DMKMON screen features to work correctly on your terminal.

ACTION

Specifies the DMKMON action to perform. Options are:

DISPLAY

Displays DMKMON reports on the terminal. The DISPLAY parameter can only be used if you are using an interactive terminal.

RECORD

Saves report calculations in a file for use at a later time. Reports are placed in the file specified on the RECORDFILE parameter.

PLAYBACK

Produces reports from previously recorded report calculations saved on a file. The file specified on the RECORDFILE parameter is the input for PLAYBACK.

LOG

Displays the reports and records report calculations for later use.

Omission of the ACTION parameter causes DISPLAY to be used if the INPUT parameter is set to a terminal. In all other cases, omission of the ACTION parameter causes RECORD to be used.

RECORDFILE

Defines the file used for recording and playing back reports. This file is defined with the standard file descriptor. Reports are stored in the file when ACTION=RECORD or ACTION=LOG is specified. Reports are read from the file when ACTION=PLAYBACK is specified.

Omission of the RECORDFILE parameter causes file DMKMONLST to be used (:family_name.user_name.DMKMONLST).

INPUT

Specifies a terminal or a sequential file of variable length records containing answers to the DMKMON prompts. In most cases, a terminal is specified on this parameter.

Omission of INPUT causes INPUT to be used.

OUTPUT

Specifies a terminal or a standard output file used by DMKMON to list error messages, prompts, and other program output. In most cases, a terminal is specified on this parameter.

Omission of the OUTPUT parameter causes $OUTPUT to be used.

VF

Specifies the valid vocabulary file. The descriptor of this file is system generated, causing the default value of VF to vary at sites that specify their own vocabulary file.

The VF parameter is optional.

Remarks For more information, see the DM Utilities Reference Manual for DM on CDC NOS/VE.

DMOPEN
IM/DM Command

Purpose Opens, optionally locks, and closes a file. If the file is busy, the utility waits until it is available.

Format

DMOPEN
FILE=dm_file_descriptor
INTENT=keyword
WAIT=keyword
LOCK=keyword
MRT=keyword
CREATE=keyword
ACCESS=keyword
REWIND=keyword
RECORDTYPE=keyword
CARRIAGE=keyword
RECORDLC=integer
BLOCKLW=integer
STATUS=status variable

Parameters *FILE*

Specifies the file to be opened. This parameter is required.

INTENT

Access required to use the file.

READ

Read access only. The file can be shared with other programs.

EXCLUSIVE

Read and write access. The file cannot be shared with other programs.

PROTECTED

Read and write access. Other programs can read, but not write to the file.

WRITE

Read and write access. The file cannot be shared with other programs.

WAIT

Indicates whether the command waits if the file is busy.

YES

Waits until the file is available.

NO

Aborts if the file is busy (another user has access to it).

If WAIT is omitted, NO is used.

LOCK

Indicates whether the file is to be locked.

YES

Locks the file so that no one but the owner can read or execute the file and no one, not even the owner, can write or delete the file.

YES requires that the user own the file or have host system privilege.

NO

The file is not locked.

If LOCK is omitted, NO is used.

MRT

Indicates whether the module revision tag is printed.

YES

The module revision tag and termination message are printed.

NO

The module revision tag and termination message are printed only at abnormal termination.

If MRT is omitted, YES is used.

CREATE

Indicates whether the file exists.

YES

File does not exist and so is created.

NO

File already exists.

If CREATE is omitted, NO is used.

ACCESS

Indicates whether the file is direct-access or sequential.

DIRECT

Direct-access file containing fixed-length records of 2048 bytes each. All DM database data files and journal files are direct-access.

SEQUENTIAL

Sequential file of fixed or variable-length records from 1 through 512 characters each. (All tape files are sequential files.)

REWIND

Indicates whether the file is repositioned at its beginning.

YES

File repositioned at its beginning.

NO

File left at its current position.

If REWIND is omitted, YES is used.

RECORDTYPE

Type of records in the file (sequential files only).

VAR

Variable-length records.

FIX

Fixed-length records.

If RECORDTYPE is omitted, VAR is used.

CARRIAGE

Indicates whether the first character of each record is a carriage control character (sequential files only).

YES

First character is carriage control.

NO

First character is data.

If CARRIAGE is omitted, YES is used.

RECORDLC

Record length in characters for sequential files (integer from 1 through 512).

If RECORDLC is omitted, 255 is used.

BLOCKLW

Block length in words for tape files (integer from 0 through 5000; 0 specifies the system default block length).

If BLOCKLW is omitted, 0 is used.

Remarks For more information, see the DM Utilities Reference Manual for DM on CDC NOS/VE.

DMPT
IM/DM Command

Purpose The DMPT command calls the DM Program Trace Record Utility which maintains program trace records created by the DMDBA program and the precompilers. By using DMPT, you can find the following:

- All the programs and subroutines that reference a particular model.
- All the views of a model, a particular program, or subroutine references.
- A variety of other combinations.

DMPT can also delete PTR records.

Format

DMPT
UID=dm_name
UPW=dm_name
DB=name
MODEL=dm_wild_name
ACTION=keyword
PROGRAM=dm_wild_name
CONFIRM=keyword
LOG=keyword
INPUT=dm_file_descriptor
OUTPUT=dm_file_descriptor
VF=dm_file_descriptor
STATUS=status variable

Parameters

UID

Specifies the user identification code. UID can be an integer or an identifier that contains letters and digits. Only the first 8 characters are used as the UID.

The UID parameter is required. DM prompts you for the UID parameter if you do not specify it on the command.

UPW

Specifies the user password. UPW can be an integer or an identifier that contains letters and digits. Only the first 8 characters are used as the UPW.

The UPW parameter is required. DM prompts you for the UPW parameter if you do not specify it on the command.

DB

Specifies the definition database containing the program trace records. DB is an identifier consisting of 1 to 8 letters or digits and must begin with a letter. The letter X, the first letter of all definition databases, need not be included.

The DB parameter is required. DM prompts you for the DB parameter if you do not specify it on the command.

MODEL

Specifies the model for which program trace records should be listed or deleted. MODEL is an identifier consisting of 1 to 8 letters or digits and must begin with a letter. You can specify an asterisk (*) as a wildcard at the end of or in place of the model name.

The MODEL parameter is required. DM prompts you for the MODEL parameter if you do not specify it on the command.

ACTION

Specifies the action you want to perform. You can specify the keywords SHOW and DELETE. SHOW means that model access information can be listed for program trace records specified by the MODEL and PROGRAM parameters. DELETE means that program trace records specified by the MODEL and PROGRAM parameters are deleted.

Omission of the ACTION parameter causes SHOW to be used.

PROGRAM

Specifies the program for which program trace records should be listed or deleted. PROGRAM is an identifier consisting of 1 to 8 letters or digits and must begin with a letter. You can specify an asterisk (*) as a wildcard at the end of or in place of the program name.

The PROGRAM parameter is required. DM prompts you for the PROGRAM parameter if you do not specify it on the command.

CONFIRM

Indicates whether or not you want the program to ask you for confirmation before deleting program trace records. You can specify the keywords YES and NO. YES means that the program asks you for confirmation before deleting each program trace record. NO means that the program deletes all the program trace records specified by the PROGRAM and MODEL parameters without asking for any confirmation.

Omission of the CONFIRM parameter causes NO to be used.

LOG

Specifies whether or not you want to be informed of the program trace records that were deleted. You can specify the keywords YES or NO. YES means that after deleting a record, the program specifies which records were deleted. NO means that the program does not specify which records were deleted.

Omission of the LOG parameter causes NO to be used.

INPUT

Specifies the terminal or a sequential file of variable length records. It contains answers to the prompts produced by the program.

Omission of the INPUT parameter causes INPUT to be used.

OUTPUT

Specifies a standard output file or the terminal. This file is used to list all error messages, prompts, and program output.

Omission of the OUTPUT parameter causes $OUTPUT to be used.

VF

Specifies a valid vocabulary file. The descriptor of this file is system generated, causing the default value of this parameter to vary at sites that specify their own vocabulary file.

Omission of the VF parameter causes the default vocabulary file of your system to be used. If your site is using the vocabulary file supplied by IM/DM, DM$VOC: is the default for the VF parameter.

Remarks For more information, see the DM Utilities Reference Manual for DM on CDC NOS/VE.

DMR
IM/DM Command

Purpose Executes the DM Restructure module to change the definition of a database object.

Format

DMR
UID=dm_name
UPW=dm_name
DB=name
ACTION=keyword
INDEX=dm_index
VERSION=dm_integer
PAD=integer

CHACEBLW=integer
INPUT=dm_file_descriptor
OUTPUT=dm_file_descriptor
VF=dm_file_descriptor
STATUS=status variable

Parameters *UID*

User identification code (only the first 8 characters are used). This parameter is required.

UPW

User password (only the first 8 characters are used). This parameter is required.

DB

Name of the database to be processed. This parameter is required.

ACTION

Action the DMR utility is to perform.

CREATE

Creates nonunique indexes that are dropped.

DROP

Drops nonunique indexes that have been created.

SHOW

Shows status of indexes.

INDEX

Record for which an index is created, specified by one of the following:

record name.element name

record name.*

.

VERSION

Version of the database to use (an integer or an asterisk). An asterisk (*) specifies all versions of the database.

A version cannot be specified when ACTION=CREATE.

If VERSION is omitted, 0 is used.

PAD

Used when ACTION=CREATE to specify the amount of padding used in references.

If PAD is omitted, 0 is used.

CHACEBLW

Used when ACTION=CREATE to specify the number of words of cache storage.

If CHACEBLW is omitted, 10000 is used.

INPUT

Input file containing any required parameters that are missing.

If INPUT is omitted, file INPUT is used.

OUTPUT

Output file on which all messages are written.

If OUTPUT is omitted, file $OUTPUT is used.

VF

Specifies a valid vocabulary file. The descriptor of this file is system generated, causing the default value of this parameter to vary at sites that specify their own vocabulary file.

Omission of the VF parameter causes the default vocabulary file of your system to be used. If your site is using the vocabulary file supplied by IM/DM, DM$VOC: is the default for the VF parameter.

Remarks For more information, see the DM Utilities Reference Manual for DM on CDC NOS/VE.

DMRW
IM/DM Command

Purpose Executes the DMRW module that initiates an IM/DM Report Writer session.

Format

DMRW
UID=dm_name
UPW=dm_name
AIDS=keyword
DB=dm_database_model
PROC=dm_file_descriptor
SIGNON=keyword
DBPROMPT=keyword
MRT=keyword
VERSION=integer
INPUT=dm_file_descriptor
OUTPUT=dm_file_descriptor
VF=dm_file_descriptor
STATUS=status variable

Parameters

UID

Specifies the user identifier. It can contain letters or digits. If omitted, DMRW prompts for the UID.

UPW

Specifies the user password. It can contain letters or digits. If omitted, DMRW prompts for the password.

AIDS

Indicates if DMRW prompts for the option to review features and terminology. The default is YES.

DB

Default database name and user data model in the form of DB=database.model. If omitted, DMRW prompts for the DB.

PROC

Specifies the file name of the command procedure automatically executed when starting a DMRW session. If omitted, DMRW does not automatically execute the procedure. However, if a login procedure exists, DM executes it whether or not you specify the PROC parameter when you log in.

SIGNON

Indicates whether DMRW is to sign on to the DM kernel (required for database access). The default is SIGNON=YES.

DBPROMPT

Specifies whether the database prompt is displayed when the DB parameter is not specified. If DBPROMPT=YES is specified and DB is not specified, the database prompt is displayed. If DBPROMPT=NO is specified, the database prompt is not displayed. This parameter is ignored if SIGNON=NO is specified. However, if DBPROMPT=NO and SIGNON=YES are both specified, the database prompt is not displayed.

MRT

Indicates whether DM displays a message on the DMRW revision number and date as the Module Revision Tag (MRT) when starting a DMRW session. If MRT=YES, DM displays the MRT. If MRT=NO, DM suppresses the MRT. The default is MRT=YES.

VERSION

Version number of the occurrence database to be referenced. Version is an integer from 0 to 99. If omitted, 0 is used.

INPUT

Specifies the terminal or the name of the sequential file of variable length records. If omitted, DMRW reads its input from INPUT.

OUTPUT

Specifies the name of the sequential file to which DMRW writes messages and prompts. If omitted, DMRW writes to the terminal.

VF

Specifies the name of the vocabulary file. If omitted, the site default vocabulary file is used.

Remarks For more information, see the DM Report Writer Reference manual.

DMSA
IM/DM Command

Purpose Executes the DM System Administration module to control who can use DM, who can create a database, and how databases are accessed.

Format

DMSA
UID=dm_name
UPW=dm_name
AIDS=keyword
INPUT=dm_file_descriptor
OUTPUT=dm_file_descriptor
VF=dm_file_descriptor
STATUS=status variable

Parameters

UID

User identification code (only the first 8 characters are used).

IM/DM prompts you for this value if it is not specified.

UPW

User password (only the first 8 characters are used).

IM/DM prompts you for this value if it is not specified.

AIDS

Indicates whether the user is given the option to review the user aids.

YES

User is asked if he wants to review the aids.

NO

User cannot review aids.

If you omit AIDS, YES is used.

INPUT

Input file containing any required parameters that are missing.

If INPUT is omitted, file INPUT is used.

OUTPUT

Output file on which all messages are written.

If OUTPUT is omitted, file $OUTPUT is used.

VF

Specifies a valid vocabulary file. The descriptor of this file is system generated, causing the default value of this parameter to vary at sites that specify their own vocabulary file.

Omission of the VF parameter causes the default vocabulary file of your system to be used. If your site is using the vocabulary file supplied by IM/DM, DM$VOC: is the default for the VF parameter.

Remarks For more information, see the DM Utilities Reference Manual for DM on CDC NOS/VE.

DMSACK
IM/DM Command

Purpose Checks referential integrity for the given database.

Format

DMSACK
UID=dm_name
UPW=dm_name
DB=dm_database_model
VERSION=integer
ASSERTIONS=list of range of dm_integer or *keyword*
ERRORLIMIT=integer
ERRORLOG=dm_file_descriptor
INPUT=dm_file_descriptor
OUTPUT=dm_file_descriptor
VF=dm_file_descriptor
STATUS=status variable

Parameters *UID*

User identification code (only the first 8 characters are used).

IM/DM prompts you for this value if it is not specified.

UPW

User password (only the first 8 characters are used).

IM/DM prompts you for this value if it is not specified.

DB

Database to be checked and the model through which the checking is done. It is specified as either

<database.model> or database.model

IM/DM prompts you for this value if it is not specified.

VERSION

Database version (0 through 99).

If VERSION is omitted, 0 is used.

ASSERTIONS or *ASSERTION*

Number of assertions to be checked.

Integer

Checks the specified assertion number.

Range of integer

Checks the specified range of assertion numbers.

ALL or *

Checks every referential constraint.

NONE

Checks assertion numbers from the INPUT file.

ERRORLIMIT

Maximum number of errors allowed (1 through 10000000). When the limit is reached, the command terminates.

If ERRORLIMIT is omitted, 1000 is used.

ERRORLOG

Reserved.

INPUT

File containing DMSACK instructions.

If INPUT is omitted, file INPUT is used.

OUTPUT

File to which all error messages, prompts, and program output is written.

If OUTPUT is omitted, the standard output file, $OUTPUT, is used.

VF

Specifies a valid vocabulary file. The descriptor of this file is system generated, causing the default value of this parameter to vary at sites that specify their own vocabulary file.

Omission of the VF parameter causes the default vocabulary file of your system to be used. If your site is using the vocabulary file supplied by IM/DM, DM$VOC: is the default for the VF parameter.

Remarks For more information, see the DM Utilities Reference Manual for DM on CDC NOS/VE.

DMSHOW
IM/DM Command

Remarks Reserved for site personnel, Control Data, or future use.

DMSPC
IM/DM Command

Remarks Reserved for site personnel, Control Data, or future use.

DMSTAT
IM/DM Command

Purpose The DMSTAT command calls the DM Display Status Message (DMSTAT) utility that explains a status code returned by a DM program.

Format

DMSTAT
MR=range of dm_integer
ANY=keyword
EXACT=keyword
VF=dm_file_descriptor
STATUS=status variable

Parameters *MR*

Specifies the message or range of messages for which the text is printed. MESSAGE_RANGE (MR) is an integer value from 1 to 3.

Omission of this parameter causes 3 to be used.

ANY

Specifies what level of error messages are printed. You can specify the keywords YES or NO. YES indicates that all levels of the messages are printed. NO means that only the first level of error messages is printed.

Omission of the ANY parameter causes NO to be used.

EXACT

Specifies whether or not error prefixes are inserted before the first line of each message. You can specify the keywords YES or NO. YES indicates that error prefixes are inserted before the first line of each message. NO indicates that error messages are not inserted before the first line of each message. The level number is printed in column 2 before each level, and Q is inserted in column 3 if $Q is encountered in the message. Column 4 contains M if MORE=YES was specified in the message definition. Also, $S causes lines to be skipped. Otherwise, $S, $P, and $Q are treated like $B. $G and $C edits are always ignored.

Omission of the EXACT parameter causes NO to be used.

VF

Specifies a valid vocabulary file. The descriptor of this file is system generated, causing the default value of this parameter to vary at sites that specify their own vocabulary file.

Omission of the VF parameter causes the default vocabulary file of your system to be used. If your site is using the vocabulary file supplied by IM/DM, DM$VOC: is the default for the VF parameter.

Remarks

- DMSTAT runs DMEMS with HEADER=NO. Thus, the same rules for message output govern DMSTAT. For further information, see the DMEMS description.
- For more information, see the DM Utilities Reference Manual for DM on CDC NOS/VE.

DMUSER
IM/DM Command

Purpose The DMUSER command calls the DM User Password and Terminal Change (DMUSER) utility that enables you to change your current password and terminal type.

Format

DMUSER
UID=dm_name
UPW=dm_name
INPUT=dm_file_descriptor
OUTPUT=dm_file_descriptor
VF=dm_file_descriptor
STATUS=status variable

Parameters *UID*

Specifies the user identification code. UID can be an integer or an identifier that contains letters and digits. Only the first 8 characters are used as the UID.

The UID parameter is required. DM prompts you for the UID parameter if you do not specify it on the command.

UPW

Specifies the user password. UPW can be an integer or an identifier that contains letters and digits. Only the first 8 characters are used as the UPW.

The UPW parameter is required. DM prompts you for the UPW parameter if you do not specify it on the command.

INPUT

Specifies the terminal or a sequential file of variable length records. It contains answers to the prompts produced by the program.

Omission of the INPUT parameter causes INPUT to be used.

OUTPUT

Specifies a standard output file or the terminal. This file is used to list all error messages, prompts, and program output.

Omission of the OUTPUT parameter causes $OUTPUT to be used.

VF

Specifies a valid vocabulary file. The descriptor of this file is system generated, causing the default value of this parameter to vary at sites that specify their own vocabulary file.

Omission of the VF parameter causes the default vocabulary file of your system to be used. If your site is using the vocabulary file supplied by IM/DM, DM$VOC: is the default for the VF parameter.

Remarks For more information, see the DM Utilities Reference Manual for DM on CDC NOS/VE.

DMVP
IM/DM Command

Purpose Executes the DM Vocabulary Processor module to maintain a vocabulary file.

Format

DMVP
ACTION=dm_question_mark or *keyword*
GET=dm_file_descriptor
OLD=dm_file_descriptor
NEW=dm_file_descriptor
MSG=integer
INPUT=dm_file_descriptor
OUTPUT=dm_file_descriptor
STATUS=status variable

Parameters *ACTION*

Action performed.

CREATE

Creates a new vocabulary file from the source statements in the GET file.

UPDATE

Creates a new vocabulary file from the source statements in the GET file and the OLD vocabulary file.

SHOW

Displays OLD vocabulary file.

MSG

Prints the text for levels 3 and 4 of a message.

If ACTION is omitted, UPDATE is used.

GET

File containing the source statements to be used.

If GET is omitted, file GET is used.

OLD

Existing vocabulary file (used with ACTION=UPDATE or SHOW).

If OLD is omitted, file OLD is used.

NEW

File to which new vocabulary file is written (used with ACTION=CREATE or UPDATE).

If NEW is omitted, file NEW is used.

MSG

Number of the message to be printed (used with ACTION=MSG).

If MSG is omitted, 0 is used.

INPUT

File containing commands for ACTION=SHOW.

If INPUT is omitted, file INPUT is used.

OUTPUT

Output file on which all messages are written.

If OUTPUT is omitted, file $OUTPUT is used.

Remarks For more information, see the DM Utilities Reference Manual for DM on CDC NOS/VE.

DMXPC
IM/DM Command

Remarks Reserved for site personnel, Control Data, or future use.

EDIT_CATALOG Command

Purpose Accesses the EDIT_CATALOG (EDIC) utility, a full screen application that can be used to create, move, copy, print, view, edit, and execute files.

Format

EDIT_CATALOG or
EDIC
CATALOG=file
DISPLAY_OPTIONS=keyword
NO_DOLLAR_FILES=boolean
STATUS=status variable

Parameters *CATALOG* or *C*

Catalog to be displayed. Omission causes the system to display the current working catalog.

DISPLAY_OPTIONS or *DISPLAY_OPTION* or *DO*

File information to be displayed.

ALL (A)

All file attributes are displayed.

BRIEF (B)

Only the name and entry type (file or catalog) are displayed.

The default is BRIEF.

NO_DOLLAR_FILES or *NDF*

Boolean indicating whether file names containing a dollar sign are to be omitted from the display. (By convention, a dollar sign character [$] appears only in CDC-defined file names.)

TRUE (ON or YES)

File names containing a $ character are not displayed.

FALSE (OFF or NO)

File names containing a $ character are displayed.

The default is FALSE.

Remarks For more information, see the NOS/VE System Usage manual.

EDIT_DECK
EDID Subcommand

Purpose Opens the specified deck in the working library for editing while maintaining your current position in other decks.

Format

EDIT_DECK or
EDID
DECK=name
STATUS=status variable

Parameters

DECK or **D**

Specifies the deck to be edited. If the deck does not exist, it is created.

This parameter is required.

Remarks

- To discard decks created unintentionally, enter:

```
end_deck write_deck=false
```

- For more information, see the NOS/VE File Editor manual.

EDIT_FILE
Command

Purpose Starts a file editor (EDIT_FILE utility) session.

Format

EDIT_FILE or
EDIF
FILE=file
INPUT=file
OUTPUT=file
PROLOG=file
DISPLAY_UNPRINTABLE_CHARACTERS=boolean
STATUS=status variable

Parameters

FILE or **F**

Specifies the name of the file you want to edit. If the file you specify does not exist, a new file is created.

The file cannot be an object file.

This parameter is required.

INPUT or *I*

Specifies the file to be used as input to the editor. This file can be positioned. This file contains optional editor subcommands used to manipulate the working file. If omitted, $COMMAND is assumed.

OUTPUT or *O*

Specifies the file to which you want to write any output that may result from your editing session. This file can be positioned.

If OUTPUT is omitted, $OUTPUT is assumed. File $OUTPUT is usually connected to the terminal.

PROLOG or *P*

Specifies the file containing subcommands you want executed each time you start the editor.

If omitted, $USER.SCU_EDITOR_PROLOG is assumed.

DISPLAY_UNPRINTABLE_CHARACTERS or *DUC*

Specifies whether unprintable ASCII characters are replaced by mnemonics when the file is displayed at the terminal. Options are:

TRUE

Unprintable characters (ASCII values 127 and 0 through 31) are replaced by their respective mnemonic values enclosed within the less than and greater than characters, < >. The mnemonics are replaced by the ASCII characters when the file is replaced.

FALSE

Unprintable characters are replaced by a single space and a warning message is issued. If the file is written when you exit the editing session, the unprintable characters are replaced by spaces.

If DISPLAY_UNPRINTABLE_CHARACTERS is omitted, FALSE is used.

ASCII characters and their corresponding mnemonic values are listed in appendix C.

Remarks

- If you would like to specify a file containing editor subcommands to be executed when you leave the editor (an epilog file), use the SET_EPILOG subcommand. If you want this done each time, include SETE in the file you specify for the PROLOG parameter.
- The following prompt appears for line editing:

  ```
  ef/
  ```

- To edit a second file while in the editor, enter the EDIT_FILE subcommand. The FILE and STATUS parameters are the only parameters allowed on the EDIT_FILE subcommand.
- For more information, see the NOS/VE File Editor manual.

Examples The following command starts the EDIT_FILE utility with file $USER.MY_FILE:

```
edit_file file=$user.my_file
```

ENTER_PPE Command

Purpose Accesses the Professional Programming Environment.

Format

ENTER_PPE or
ENTPPE or
ENTP
ENVIRONMENT_CATALOG=file
STATUS=status variable

Parameters *ENVIRONMENT_CATALOG* or *EC*

Path to the subcatalog for which PPE is executed. It is the lowest level of the PPE environment catalog hierarchy presented in the current session.

PPE creates the subcatalog if it does not exist. If the subcatalog belongs to another user, the owner must grant you the following catalog permit:

```
access_modes=(all, cycle, control)
application_information='I1'
```

If you omit ENVIRONMENT_CATALOG, the subcatalog used is $USER.PROFESSIONAL_ENVIRONMENT.

Remarks

- The Professional Programming Environment uses the full-screen interface.
- For more information, see the online ENVIRONMENT manual.

Examples

The following command starts a PPE session in which the PPE level is the default subcatalog, $USER.PROFESSIONAL_ENVIRONMENT.

```
/entp
```

The following command starts a PPE session in which the PPE level is the subcatalog named $USER.XYZ.PPE_WORK.

```
/entp $user.xyz.ppe_work
```

ENTER_PROGRAMMING_ENVIRONMENT Command

Purpose

Accesses the Programming Environment.

Format

ENTER_PROGRAMMING_ENVIRONMENT or **ENTPE**

DEFAULT_PROCESSOR=keyword
ENVIRONMENT_CATALOG=file
STATUS=status variable

Parameters

DEFAULT_PROCESSOR or *DP*

Language in which you intend to program (C, COBOL, CORAL, FORTRAN Version 1, FORTRAN Version 2, or Pascal). Default is FORTRAN.

ENVIRONMENT_CATALOG or *EC*

Catalog in which the Programming Environment is to maintain its files. Default is $USER.PROGRAMMING_ENVIRONMENT.

Remarks

- You can access the ENVIRONMENT online manual from within or outside of the utility using the command:

  /explain m=environment

- The Programming Environment uses the full screen interface.

- For more information, see the online ENVIRONMENT manual.

Examples

The following command enters the Programming Environment.

```
/enter_programming_environment
```

The next command enters the Programming Environment with COBOL as the the default processor.

```
/entpe default_processor=cobol
```

The third command enters the environment maintained in the $USER.BUSINESS catalog.

```
/entpe environment_catalog=$user.business
```

EXECUTE_COMMAND Command

Purpose

Executes a single command asynchronously in a new task. Utility subcommands cannot be executed using this command.

Format

EXECUTE_COMMAND or
EXEC
COMMAND=string
TASK_NAME=name
COMMAND_FILE=file
ENABLE_ECHOING=boolean
STATUS=status variable

Parameters

COMMAND or **C**

Specifies the command to be executed. The text of this parameter must conform to the syntax requirements for commands. This parameter is required.

TASK_NAME or *TN*

Specifies a name used to refer to the task.

COMMAND_FILE or *CF*

Specifies a file, a copy of which becomes the current command file (such as $COMMAND) within the new task. This parameter is only necessary if the command being executed is a command utility, or references the current command file.

Omission causes no current command file to be defined for the new task.

ENABLE_ECHOING or *ENABLE_ECHO* or *EE*

Specifies whether the command is to be echoed back to the terminal. Values can be:

YES

Echoing enabled

NO

Echoing not enabled

Remarks For more information, see the NOS/VE System Usage manual.

EXECUTE_TASK Command

Purpose Executes the program described on the command.

Format

EXECUTE_TASK or
EXET
FILES=list of file
PARAMETERS=string
LIBRARIES=list of file or *keyword*
MODULES=list of name
STARTING_PROCEDURE=any
LOAD_MAP=file
LOAD_MAP_OPTIONS=list of keyword
PRESET_VALUE=keyword
TERMINATION_ERROR_LEVEL=keyword
STACK_SIZE=integer
DEBUG_INPUT=file
DEBUG_OUTPUT=file
ABORT_FILE=file
DEBUG_MODE=boolean
TASK_NAME=name

ARITHMETIC_OVERFLOW = boolean
ARITHMETIC_LOSS_OF_SIGNIFICANCE = boolean
DIVIDE_FAULT = boolean
EXPONENT_OVERFLOW = boolean
EXPONENT_UNDERFLOW = boolean
FP_INDEFINITE = boolean
FP_LOSS_OF_SIGNIFICANCE = boolean
INVALID_BDP_DATA = boolean
STATUS = status variable

Parameters

FILES or *FILE* or *F*

Object list. Optional list of object files or object library files whose modules are unconditionally loaded.

If FILE is omitted, the modules executed are determined by the MODULE and STARTING_PROCEDURE parameters. If the FILE, MODULE, and STARTING_PROCEDURE parameters are all omitted, the loader attempts to execute the modules on file $LOCAL.LGO.

PARAMETERS or *PARAMETER* or *P*

String passed to the program as its parameter list.

LIBRARIES or *LIBRARY* or *L*

List of object libraries added to the beginning of the program library list. If LIBRARY is omitted, the program library list consists of the local library list, the NOS/VE task services library, text embedded libraries, the job library list, and the job debug library list if DEBUG_MODE is ON.

MODULES or *MODULE* or *M*

Module list. Optional list of modules unconditionally loaded from object libraries in the program library list.

You use a string value for a module whose name is not an SCL name. Some examples of such module names are: a COBOL module, where a hyphen character (-) may be part of the name, and a C function, where lower case is significant.

If MODULE is omitted, the modules executed are determined by the FILE and STARTING_PROCEDURE parameters. If the FILE, MODULE, and STARTING_PROCEDURE parameters are all omitted, the loader attempts to execute the modules on object file $LOCAL.LGO.

STARTING_PROCEDURE or *SP*

Name of the entry point where execution begins.

You use a string value for an entry point whose name is not an SCL name.

If STARTING_PROCEDURE is omitted, the last transfer symbol loaded is used.

LOAD_MAP or *LM*

File on which the load map is written. This file can be positioned. If LOAD_MAP is omitted, the job default value is used. You can display the job default value with the DISPLAY_PROGRAM_ATTRIBUTES command.

LOAD_MAP_OPTIONS or *LOAD_MAP_OPTION* or *LMO*

Set of one or more keywords indicating the information included in the load map. Options are:

NONE

No load map is written.

SEGMENT (S)

Segment map.

BLOCK (B)

Block map.

ENTRY_POINT (EP)

Entry point map.

CROSS_REFERENCE (CR)

Entry point cross-reference.

ALL

Segment map, block map, entry point map, and entry point cross-reference.

If LOAD_MAP_OPTION is omitted, the job default load map option is used. You can display the job default value with the DISPLAY_PROGRAM_ATTRIBUTES command.

PRESET_VALUE or *PV*

Value stored in all uninitialized words of program space. Options are:

ZERO (Z)

All zeros.

FLOATING_POINT_INDEFINITE (FPI)

Floating-point indefinite value.

INFINITY (I)

Floating-point infinite value.

ALTERNATE_ONES (AO)

Alternating 0 and 1 bits; the leftmost (highest order) bit is 1.

If PRESET_VALUE is omitted, the job default value is used. You can display the job default value with the DISPLAY_PROGRAM_ATTRIBUTES command.

TERMINATION_ERROR_LEVEL or *TEL*

Error level that terminates program loading. Options are:

WARNING (W)

Warning, error, or fatal error.

ERROR (E)

Error or fatal error only.

FATAL (F)

Fatal error only.

If TERMINATION_ERROR_LEVEL is omitted, the job default termination error level is used. You can display the job default value with a DISPLAY_PROGRAM_ATTRIBUTES command.

STACK_SIZE or *SS*

Maximum number of bytes in the run-time stack. The program uses the run-time stack for procedure call linkages and local variables. If STACK_SIZE is omitted, the system default value is used. You can display the default value with a DISPLAY_PROGRAM_ATTRIBUTES command.

DEBUG_INPUT or *DI*

File containing Debug commands. The commands are read only if the program is executed in Debug mode. This file can be positioned. If DEBUG_INPUT is omitted, the default Debug input file for the job is used. You can display the job default value with the DISPLAY_PROGRAM_ATTRIBUTES command.

DEBUG_OUTPUT or *DO*

File on which Debug output is written. Output is written only if the program is executed in Debug mode. This file can be positioned. If DEBUG_OUTPUT is omitted, the default Debug output file for the job is used. You can display the job default value with the DISPLAY_PROGRAM_ATTRIBUTES command.

ABORT_FILE or *AF*

File containing Debug commands to be processed if the program aborts. The commands are used only if the program is not executed in Debug mode. This file can be positioned.

If ABORT_FILE is omitted, the job default abort file is used. You can display the job default value with a DISPLAY_PROGRAM_ATTRIBUTES command.

DEBUG_MODE or *DM*

Indicates whether the program is to be run in Debug mode. (For information on using Debug, refer to the specific program's source language manual.) Options are:

ON

Program executed under Debug control.

OFF

Program executed without Debug control.

If DEBUG_MODE is omitted, the job default value is used. You can display the job default value with the DISPLAY_PROGRAM_ATTRIBUTES command.

TASK_NAME or *TN*

This parameter specifies that the program is to be executed as an asynchronous task and provides a name that can be used to refer to the task.

Omission causes the program to be executed synchronously.

ARITHMETIC_OVERFLOW or *AO*

This parameter specifies whether or not the hardware condition ARITHMETIC_OVERFLOW causes an interrupt. Valid specifications are:

ON

ARITHMETIC_OVERFLOW is enabled. The condition causes an interrupt.

OFF

ARITHMETIC_OVERFLOW is disabled. The condition does not cause an interrupt.

ARITHMETIC_LOSS_OF_SIGNIFICANCE or *ALOS*

This parameter specifies whether or not the hardware condition ARITHMETIC_LOSS_OF_SIGNIFICANCE causes an interrupt. Valid specifications are:

ON

ARITHMETIC_LOSS_OF_SIGNIFICANCE is enabled. The condition causes an interrupt.

OFF

ARITHMETIC_LOSS_OF_SIGNIFICANCE is disabled. The condition does not cause an interrupt.

DIVIDE_FAULT or *DF*

This parameter specifies whether or not the hardware condition DIVIDE_FAULT causes an interrupt. Valid specifications are:

ON

DIVIDE_FAULT is enabled. The condition causes an interrupt.

OFF

DIVIDE_FAULT is disabled. The condition does not cause an interrupt.

EXPONENT_OVERFLOW or *EO*

This parameter specifies whether or not the hardware condition EXPONENT_OVERFLOW causes an interrupt. Valid specifications are:

ON

EXPONENT_OVERFLOW is enabled. The condition causes an interrupt.

OFF

EXPONENT_OVERFLOW is disabled. The condition does not cause an interrupt.

EXPONENT_UNDERFLOW or *EU*

This parameter specifies whether or not the hardware condition EXPONENT_UNDERFLOW causes an interrupt. Valid specifications are:

ON

EXPONENT_UNDERFLOW is enabled. The condition causes an interrupt.

OFF

EXPONENT_UNDERFLOW is disabled. The condition does not cause an interrupt.

FP_INDEFINITE or *FPI* or *FI*

This parameter specifies whether or not the hardware condition FP_INDEFINITE causes an interrupt. Valid specifications are:

ON

FP_INDEFINITE is enabled. The condition causes an interrupt.

OFF

FP_INDEFINITE is disabled. The condition does not cause an interrupt.

FP_LOSS_OF_SIGNIFICANCE or *FPLOS* or *FLOS*

This parameter specifies whether or not the hardware condition FP_LOSS_OF_SIGNIFICANCE causes an interrupt. Valid specifications are:

ON

FP_LOSS_OF_SIGNIFICANCE is enabled. The condition causes an interrupt.

OFF

FP_LOSS_OF_SIGNIFICANCE is disabled. The condition does not cause an interrupt.

INVALID_BDP_DATA or *IBDPD* or *IBD*

This parameter specifies whether or not the hardware condition INVALID_BDP_DATA causes an interrupt. Valid specifications are:

ON

INVALID_BDP_DATA is enabled. The condition causes an interrupt.

OFF

INVALID_BDP_DATA is disabled. The condition does not cause an interrupt.

Remarks

- The FILE, MODULE, and STARTING_PROCEDURE parameters specify the modules executed. If you omit all three parameters, the command attempts to execute object file $LOCAL.LGO.
- If you specify a string using the PARAMETER parameter, the string is passed to the program as its parameter list.
- You can specify the program to be executed and any of its execution options on the EXECUTE_TASK command or use default values. You can display the default execution options with a DISPLAY_PROGRAM_ATTRIBUTES command and change them with a SET_PROGRAM_ATTRIBUTES command.
- For more information, see the NOS/VE Object Code Management manual.

Examples

- The following command executes the object modules on files OBJ1 and OBJ2.

```
/execute_task (obj1,obj2)
```

- The following command executes the object modules on file OBJ1 in Debug mode. Assuming the initial default values have not been changed, Debug commands are read from file COMMAND and Debug output is written on file $OUTPUT.

```
/execute_task obj1 debug_mode=on
```

EXIT Control Statement

Purpose

Transfers control out of a structured statement, command procedure, or command utility.

Format

EXIT
designator
WHEN boolean expression
WITH status expression

The following are valid formats of the EXIT statement:

EXIT
EXIT *designator*
EXIT *WHEN boolean expression*
EXIT *designator WHEN boolean expression*
EXIT *designator WITH status expression*
EXIT *WHEN boolean expression WITH status expression*
EXIT *designator WHEN boolean expression WITH status expression*

Parameters

designator

Designates the labeled structured statement, command procedure or command utility to be exited.

label

Label associated with the enclosing structured statement. If you specify a label, the enclosing structured statement with that label is exited. If you omit a label, the innermost structured statement is exited.

You may not use the WITH clause with structured statements.

procedure name

Name of an active procedure. This name must be the first in the list of names in the procedure header. To exit the enclosing procedure, use PROCEDURE or PROC.

utility name

Name of an active utility. To exit the innermost active utility, use UTILITY.

boolean expression

Specifiies whether the designated enclosing statement should be exited. If the expression is TRUE, the statement is exited. If the expression is FALSE, or if it is omitted, the statement is not exited.

status expression

Specification indicating the status condition under which an exit is to take place. This expression must be a procedure's status value. This status becomes the termination status of the specified procedure. If you omit the WITH clause when exiting a procedure or utility, termination with normal status occurs.

Remarks

- The WHEN clause is evaluated at the point of reference rather than continually, as is done by the WHEN condition handler (discussed in the SCL Language Definition manual).
- You can use the WHEN and WITH clauses either separately or together. If used together, you can list either clause first.
- Only enclosing structured statements that are self-contained can be exited. For example, only structured statements with no outstanding file references can be exited.
- Any active command procedure or utility can be exited.
- For more information, see the NOS/VE System Usage manual.

Examples

- In the block structure example that follows, 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.

```
            block1: loop
                    .
|-------------- exit      First EXIT statement
|                   .
|                   block2: loop
|                           .
|   |-------------- exit block1 Second EXIT statement
|   |               .
|   |   +---------- exit Third EXIT statement
|   |   |           .
|   |   |       loopend block2
|   |   |
|   |   |------>next statement
|   |
|   |       loopend block1
|   |
|---+------>next statement
```

- 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-- CI=US CC=1 TEXT=?Proc Example Failed
```

EXIT_PROC
Control Statement

Purpose Exits a procedure. Execution resumes at the statement following the procedure call.

Format

EXIT_PROC
WITH status expression
WHEN boolean expression

The following are valid formats of the EXIT_PROC statement:

EXIT_PROC
EXIT_PROC*WITH status expression*
EXIT_PROC*WHEN boolean expression*

Parameters *status expression*

Specifies the status to be returned to the block from which the procedure was called. If you omit this parameter, a normal status is returned.

boolean expression

Specifies whether exiting from the designated procedure should take place. If the expression is TRUE or omitted, the exit is performed. If the expression is FALSE, the exit does not take place.

Remarks

- A status parameter need not be included in the procedure definition to use the WITH clause.
- The WHEN clause is evaluated at the point of reference rather than continually, as is done by the WHEN condition handler.
- For more information, see the NOS/VE System Usage manual.

Examples The following example exits a procedure with an abnormal status.

```
PROC example (status)
    .
    .
  EXIT_PROC WITH ..
    $status(false,'US',1,'Proc Example Failed')
    .
    .
PROCEND example
```

If the EXIT_PROC statement is executed, the following output is returned.

```
/example
 --ERROR-- CI=US CC=1 TEXT=?Proc Example Failed
```

EXPAND_SOURCE_FILE Command

Purpose Expands a text file as though the file were a deck on an SCU library. Expanding a file processes the directives embedded in the source text and copies the expanded text to a separate compile file.

Format

EXPAND_SOURCE_FILE or
EXPSF
FILE=file
COMPILE=file
SELECTION_CRITERIA=file
WIDTH=integer
LINE_IDENTIFIER=keyword
ALTERNATE_BASE=list of file
LIST=file
EXPANSION_DEPTH=integer
DISPLAY_OPTIONS=keyword
STATUS=status variable

Parameters

FILE or **F**

File to be expanded. This parameter is required.

COMPILE or *C*

File on which the expanded text is written. You can specify a file position as part of the file name. If COMPILE is omitted, file COMPILE is used.

SELECTION_CRITERIA or *SC*

File from which selection criteria subcommands are read. You can specify a file position as part of the file name. To enter selection criteria subcommands interactively, specify COMMAND. If SELECTION_CRITERIA is omitted, no selection criteria processing is performed.

WIDTH or *W*

Length of the expanded lines, excluding line identifiers. If WIDTH is omitted, the default line width is 0 (zero).

LINE_IDENTIFIER or *LI*

Line identifier placement.

RIGHT (R)

Line identifiers are placed to the right of the text.

LEFT (L)

Line identifiers are placed to the left of the text.

NONE

No line identifiers are placed on output lines.

If LINE_IDENTIFIER is omitted, NONE is used.

ALTERNATE_BASE or *ALTERNATE_BASES* or *AB*

Optional list of one or more additional libraries to be searched for decks.

LIST or *L*

Listing file. You can specify a file position as part of the file name. If LIST is omitted, the listing file is the file specified on the SET_LIST_OPTIONS subcommand. Otherwise, the default is file $LIST.

EXPANSION_DEPTH or *ED*

Number of levels of COPY and COPYC directives to process. COPY and COPYC directives beyond the maximum expansion depth are expanded as text. If EXPANSION_DEPTH is omitted, COPY and COPYC directives are processed whenever they are encountered.

DISPLAY_OPTIONS or *DO*

Indicates whether the listing includes the library for each deck from which the deck was expanded.

BRIEF (B)

Does not list the decks or their library origins.

FULL (F)

Lists the library origin when more than one library is used.

If DISPLAY_OPTIONS is omitted, BRIEF is used.

Remarks

- EXPAND_SOURCE_FILE allows you to expand a text file without starting an SCU session. It is identical to the SCU subcommand EXPAND_FILE, except that the EXPAND_SOURCE_FILE command does not interact with the working library. Although the command can be entered within an SCU session, it has no effect on the working library of the session. It can be used to expand files outside of an SCU session.

- You can specify alternate base libraries with the ALTERNATE_BASE parameter. When SCU processes a COPY or COPYC directive, it searches the deck lists of the alternative base libraries in the order the libraries are listed on the ALTERNATE_BASE parameter.

- The EXPANSION_DEPTH parameter can limit the levels of nested directives processed. If SCU reads a directive at a level beyond the maximum level processed, it expands it as text.

- The LINE_IDENTIFIER and WIDTH parameters affect how the expanded text is written on the compile file.

- The line width can be specified by the WIDTH parameter. If the line width for a file or deck is 0 (zero), EXPAND_SOURCE_FILE writes each line as it is stored in the file or deck (no trailing blanks or truncation); a blank line, therefore, is written as a zero-length V record. If the line width for a file or a

deck is nonzero, EXPAND_SOURCE_FILE writes each line using that width. Lines shorter than the width are padded with trailing blanks; lines longer than the width are truncated.

- For more information, see the NOS/VE Source Code Management manual.

Examples The following command expands the text of file OLD_TEXT and writes the expanded text on file COMPILE. The unique name given to the temporary deck created from file OLD_TEXT is $95 .. 28.

```
/expand_source_file old_text alternate_base=source_library ..
../display_options=full list=output
*=Deck was copied
 $80071613250209D19880225T220933 Working Library
* SOURCE_LIBRARY                 F$$__00011BD0_E3
```

EXPLAIN Command

Purpose Displays the text of an online manual.

Format

EXPLAIN or
EXP
SUBJECT=string
MANUAL=file
LIST=file
EXPAND_DEPTH=integer
STATUS=status variable
$CHILD=name

Parameters

SUBJECT or *S*

Specifies the initial index topic to be used to locate information. Omission causes the main menu of the manual to be displayed.

MANUAL or *M*

Specifies the online manual that is to be displayed. Omission causes NOS_VE, the name of the default online manual, to be used.

LIST or *L*

Identifies the file to receive listable output produced by the EXCERPT directive or the COPY key. Omission causes file MANUAL_PAGES to be used.

EXPAND_DEPTH or *ED*

Specifies the number of levels of topics to be initially displayed if you are using a TOPICS online manual. You may specify a value of 1 to 10.

If omitted, 1 is assumed.

$CHILD

Reserved.

Remarks

- If your interaction style is SCREEN, EXPLAIN uses screen mode to display the on-line manual.
- EXPLAIN will first attempt to access the file using the name specified by the MANUAL parameter. If no such file exists in the specified catalog, then $SYSTEM.MANUALS.NOS_VE will be prefixed and another attempt to access the manual will be made.
- For more information, see the NOS/VE System Usage manual.

Examples

In the following example, the EXPLAIN command is entered without parameters, causing the default online manual to be displayed. The default online manual can also be accessed by specifying file $SYSTEM.MANUALS.NOS_VE on the MANUAL parameter.

```
/explain
```

The following example calls the online FORTRAN manual directly.

```
/explain m=fortran
```

The following example displays the description of the ACCEPT_LINE command from the online SCL manual.

```
/explain 'accept_line' scl
```

EXPLAIN_MESSAGE Command

Purpose Requests a description of a system message.

Format

EXPLAIN_MESSAGE or
EXPM
CONDITION=integer
IDENTIFIER=string
STATUS=status variable
CLV$PREVIOUS_STATUS=status

Parameters

CONDITION or *C*

Specifies a condition code identifying a message for which an explanation is desired. Omission causes the code for the last response message you received to be used. If the previous status is normal, the first screen of the default online manual is displayed.

IDENTIFIER or *ID* or *I*

Two character product identifier associated with the condition code.

CLV$PREVIOUS_STATUS

Reserved.

Remarks

- This explanation is more detailed than that provided in a BRIEF or FULL mode message.
- Refer to the CHANGE_MESSAGE_LEVEL command for further information.
- For more information, see the NOS/VE System Usage manual.

Examples Assume you receive the following system message.

```
/attach_file not_a_file
--ERROR--File "NOT_A_FILE" does not exist or you are
  not permitted for any access.
```

Entering HELP or EXPLAIN_MESSAGE without parameters causes a description of the most recent message to be displayed.

In the following example, a message description is obtained by specifying the condition code on the EXPLAIN_MESSAGE command.

```
/explain_message c=1016 id='am'
```

EXTRACT_SOURCE_LIBRARY Command

Purpose Extracts a set of decks from the base library for use as a separate library.

Format

EXTRACT_SOURCE_LIBRARY or
EXTSL
DECK=list of range of name
INTERLOCK=name
SELECTION_CRITERIA=file
BASE=file
RESULT=file
STATUS=status variable

Parameters

DECK or *DECKS* or *D*

Decks to be copied. The decks can be specified as a list of one or more names, a list of one or more ranges, or as the keyword ALL. ALL specifies all decks on the base library. If DECK is omitted, the decks copied are determined by the contents of the criteria file.

INTERLOCK or **I**

Name of the user reserving the extracted decks; use the keyword NONE if the decks are not to be reserved. The name is written in the subinterlock field for each extracted deck on the base library and in the original interlock field of each deck in the extracted library. This parameter is required.

SELECTION_CRITERIA or *SC*

File from which selection criteria commands are read. You can specify a file position as part of the file name. If SELECTION_CRITERIA is omitted, decks are selected using the DECK parameter.

NOTE

If an interlock is to be set, you cannot use the selection criteria commands EXCLUDE_MODIFICATION, EXCLUDE_FEATURE, or EXCLUDE_STATE to exclude modifications. Interlocked decks can only be extracted as a whole.

BASE or *B*

File containing the source library from which decks are copied. If BASE is omitted, file SOURCE_LIBRARY in your current working catalog is used.

If the EXTRACT_SOURCE_LIBRARY command sets interlocks, it modifies the base library file by writing the interlock value in the original interlock field of the deck header of each extracted deck.

RESULT or **R**

File on which the new source library is written. This parameter is required.

Remarks

- The EXTRACT_SOURCE_LIBRARY command is a NOS/VE command. Although you can enter the command during an SCU session, it has no effect on the working library of the session. However, if both use the same result file, the first file is overwritten by the second.

  To set interlocks with an EXTRACT_SOURCE_LIBRARY command, you must have modify permission as well as read permission to the base library file. You also must have interlock authority for the file (the letter I in the application information field of your file permit entry).

- If you intend to later merge the extracted library decks with the base library decks to form a new library, you can set interlocks on the extracted decks to notify other users of the base library that you have extracted the decks. You can set interlocks in the extracted decks by specifying a user name on the INTERLOCK parameter.

- When setting interlocks, the command stores the user name both in the deck header of the extracted deck copy and in the deck header of the original deck. The name is stored in the original interlock field of the extracted deck copy and in the subinterlock field of the original deck.

- If you set interlocks when you extracted the library, the REPLACE_LIBRARY or COMBINE_LIBRARY subcommand enforces the interlock if you specify ENFORCE_INTERLOCKS=TRUE on the subcommand. Interlock enforcement means that REPLACE_LIBRARY or COMBINE_LIBRARY checks whether the original interlock value in the header of the extracted deck copy matches the subinterlock value in the header of the working library copy.

  If the values match, REPLACE_LIBRARY or COMBINE_LIBRARY replaces the working library deck with the extracted deck; otherwise, it issues a warning message, does not replace the working library deck, and attempts replacement of any remaining decks in the deck list.

- The key characters of source libraries must match.

- You can select the decks extracted by deck names, selection criteria, or names qualified by selection criteria. SCU begins with the decks specified on the DECK parameter and then adds and removes decks as specified by selection criteria commands.

- The modification, feature, and group lists for the extracted library contain only the modifications, features, and groups applicable to the extracted decks.

- For more information, see the NOS/VE Source Code Management manual.

Examples

The following command copies the deck DECK1 and the decks in the range DECK5 through DECK7 from the base library on permanent file OLDPL to the result library on permanent file NEWLIB. No interlocks are set.

```
/extract_source_library (deck1,deck5..deck7) ..
../interlock=none base=$user.oldpl result=$user.newlib
```

$FILE Function

Purpose Returns certain file attributes.

Format

```
$FILE
   (file
   keyword)
```

Parameters

file

Specifies the name of the file whose attributes you are querying. This parameter is required.

keyword

Specifies the file attribute you are querying. Chapter 3, Function Attributes, lists and describes the keyword values you can supply and the corresponding function results. This parameter is required.

Remarks

- Depending on the attribute being tested, either boolean, string, or integer results are possible. When a string value is returned, all letters within the string are converted to uppercase (the exception is the USER_INFORMATION attribute).
- For further information about functions, see the NOS/VE System Usage manual.

Examples

- The following example queries whether a file is temporary. If it is temporary, the statements in the IF sequence are executed; otherwise, they are skipped.

```
IF $file(data_file_1,temporary) THEN
     .
     .
   "Process statement list
   "if DATA_FILE_1 is temporary.
IFEND
```

  When the keyword TEMPORARY is used, $FILE returns a TRUE value only if the specified file exists.

- The following example queries whether the FILE_ORGANIZATION attribute is SEQUENTIAL and the FILE_CONTENT attribute is LEGIBLE. If this is the case, the file is compiled.

```
IF $file(program_able,file_organization) ..
  = 'SEQUENTIAL' and ..
  $file(program_able,file_content) ..
  = 'LEGIBLE' THEN
  fortran i=program_able
IFEND
```

- The following example extracts the USER_INFORMATION file attribute and displays the string value:

```
/display_value $file(data_file_1,user_information)
Data file for CONV_USER program.
```

FILE_MANAGEMENT_UTILITY Command

Purpose Executes the File Management Utility.

Format

FILE_MANAGEMENT_UTILITY or
FILMU or
FMU
INPUT=file
OUTPUT=file
DIRECTIVES=file
LIST=file
ERROR_DISPOSITION=keyword
STATUS=status variable

Parameters

INPUT or *I*

File reference of file to be copied. Specified only if DIRECTIVES is omitted. Default is $INPUT.

OUTPUT or *O*

File reference of file to be copied to. Specified only if DIRECTIVES is omitted. Default is $OUTPUT.

DIRECTIVES or *DIRECTIVE* or *DIR* or *D*

File reference of a file containing FMU directives. Specified only if INPUT and OUTPUT are omitted. Required if INPUT and OUTPUT are omitted.

LIST or *L*

File reference of the file containing the summary of the FMU run, including diagnostic messages. Required in an interactive session. Default is $LIST.

ERROR_DISPOSITION or *ED*

Indicates whether FMU is to abort if any output file is closed prematurely due to an error.

ABORT (A)

FMU aborts (default).

NO_ABORT (NA)

FMU continues writing the other output files.

Remarks

- FMU performs a simple copy when the INPUT and OUTPUT parameters are specified. When INPUT and OUTPUT are omitted, FMU reads directives from the file specified by the DIRECTIVES parameter. The directives specify the input and output files and the data reformatting to be preformed.
- For more information, see the NOS/VE Advanced File Management Usage manual.

Examples

FMU can be used to do a simple file copy.

```
/FMU,INPUT=IN_FILE,OUTPUT=OUT_FILE,LIST=FMULIST,ED=NA
```

FMU can be used to perform data reformatting:

```
/FMU,DIR=DIRFILE,LIST=FMLIST
```

FIXC2 IM/DM Command

Remarks Reserved for site personnel, Control Data, or future use.

$FNAME Function

Purpose Converts a string to a file name.

Format

$FNAME
(string)

Parameters **string**

Specifies the string variable you want converted to a file name. This parameter is required.

Remarks

- This function is useful in constructing a file name from a string value and using the file name as a parameter on another command or function.
- For further information about functions, see the NOS/VE System Usage manual.

Examples The following example uses the $FNAME function result as a file name parameter.

```
/catalog_name = '$user'
/file_name = 'data_file_1'
/attach_file $fname(catalog_name//'.'//file_name)
```

In the preceding example, a variable CATALOG_NAME is created with the string value '$user'. Similarly, a variable FILE_NAME is created with the string value 'data_file_1'. The string expression is passed to the $FNAME function, which is evaluated as the following:

```
$user.data_file_1
```

Subsequently, the $FNAME function converts the evaluated string to the file name $USER.DATA_FILE_1, and the ATTACH_FILE command is processed with the parameter converted as follows:

```
ATTACH_FILE $USER.DATA_FILE_1
```

FOR
Control Statement

Purpose Provides controlled repetition of a statement list.

Format

label: **FOR**
variable=initial TO final *BY step* **DO**
statement list
FOREND *label*

Parameters

label

Specifies the name of the FOR block. This label can be used by CYCLE or EXIT statements within the block.

variable

Specifies the control variable of the FOR statement. If not already declared, it is created by the FOR statement, in which case the variable may not contain a subscript or field qualifier. This parameter is required.

initial

Integer that specifies the initial value of the control variable. This parameter is required.

final

Integer expression that specifies the final value of the control variable. This parameter is required.

step

Integer expression that specifies the increment of the control variable for each loop. The default increment is 1. If the step element is omitted, the FOR statement has the following format:

```
label: FOR variable = initial TO final DO
```

statement list

Specifies the list of statements that reside in the block.

Remarks

- Any alteration of the control variable or of the initial, final, or step elements within the statement list has no effect on the number of iterations of the FOR statement.
- If initial is less than final, step must be positive or the FOR statement is not executed.
- If initial is greater than final, step must be negative or the FOR statement is not executed.
- If initial equals final or step is 0 (zero), the statement list is executed once.

- If the control variable is altered and then referenced within the statement list, the altered value is used. The control variable, however, is reset to the correct value for the next iteration of the FOR statement. After exiting from the FOR statement, the control variable has the last value that it attained, either due to alteration or FOR statement iteration.

- For more information, see the NOS/VE System Usage manual.

Examples The following example displays each character in a string on a separate line.

```
/string = 'down hill'
/blanks = ' '
/for i = 1 to $strlen(string) do
for/display_value ..
for../$substr(blanks,1,i)//$substr(string,i,1)
for/forend
 d
  o
   w
    n

      h
       i
        l
         l
```

FORMAT_CYBIL_SOURCE Command

Purpose Formats CYBIL source code for consistency and greater readability.

Format

FORMAT_CYBIL_SOURCE or
FORCS
INPUT=file
OUTPUT=file
ERROR=file
FORMAT_OPTIONS=list of keyword
LINE_WIDTH=integer
KEY=string
STATUS=status variable

Parameters

INPUT or *I*

Specifies the file from which the CYBIL source code is read. If omitted, $INPUT is assumed.

OUTPUT or *O*

Specifies the file on which the formatted CYBIL source code is written. If omitted, $OUTPUT is assumed.

ERROR or *E*

Specifies the file on which error messages are written. If omitted, $ERRORS is assumed.

FORMAT_OPTIONS or *FO*

Specifies one or more of the following format options. If omitted, NONE (no format options are selected) is assumed.

ALIGN_COMMENTS or AC — Specifies that comment blocks are aligned in column one as follows:

- Comment blocks not beginning in column one are repositioned to column one.
- Comment lines exceeding the line width are truncated and the text aligned with the first non-blank character of the original comment line.
- Trailing comment delimiters are removed so that comments end with the end-of-line.

ALL — Selects all the format options (CB, ME, and NC).

COMMENT_BLOCK or CB — Specifies that a comment block is preceded and followed by a blank line. A comment block is considered to be one or more lines of comments.

MARK_ EXIT or ME	Marks exit statements (that is, EXIT, CYCLE, and RETURN) by adding the comment {----->* after the statement. This indicates that a change in the program's flow of control occurs here. If the comment delimiter (the left brace) is already in the statement, the statement is not changed.
NO_ COMPRESS or NC	Selects no compression of successive space characters. The same number of space characters in the input file are written to the output file. This option is overridden if the formatter finds an FMT directive in the source code with the COMPRESS parameter set to ON. Unless otherwise specified, spaces are compressed.
NONE	No format options are selected.

LINE_WIDTH or *LW*

Specifies the line width of the formatted output. You can specify an integer from 11 to 110. If omitted, a right margin of 79 is assumed.

This setting is overridden if the formatter finds the LEFT and RIGHT layout control directives in the source code.

KEY or *K*

Specifies the key character that indicates embedded Source Code Utility directives. This value is specified as a 1-character string. Statements that begin with the key character in column 1 are not formatted. If omitted, the asterisk character is assumed.

Remarks

- The file paths specified for the input and output files cannot be the same. For example, INPUT=PROC1 and OUTPUT=PROC1 are not valid; INPUT=$LOCAL.PROC1 and OUTPUT=$USER.PROC1 however, are valid.

- The input and output files are rewound before formatting if you don't specify a file position in the file reference.

- For more information, see the CYBIL Language Definition manual.

Examples The following command formats the CYBIL source program contained on file INITIAL and writes it to $USER.FINAL.

```
/format_cybil_source initial $user.final
```

FORMAT_SCL_PROC Command

Purpose Formats a file containing SCL commands so that the procedure is easier to read.

Format

FORMAT_SCL_PROC or
FORMAT_SCL_PROCS or
FORSP or
FORSCLP
INPUT=file
OUTPUT=file
PAGE_WIDTH=integer
INITIAL_INDENT_COLUMN=integer
KEY_CHARACTER=string
UTILITY_DEFINITION_FILE=file
PROCESS_COLLECT_TEXT=boolean
STATUS=status variable

Parameters

INPUT or I

Specifies the name of the file from which the commands are read. This file must have the following file attribute values:

- The FILE_CONTENTS attribute must be either LEGIBLE or UNKNOWN.

- The FILE_PROCESSOR attribute must be either SCL or UNKNOWN.

- The FILE_STRUCTURE attribute must be either DATA or UNKNOWN.

This parameter is required.

OUTPUT or **O**

Specifies the name of the file to which the formatted commands are written. This file must have the same attribute values as those described for the INPUT file. This parameter is required.

PAGE_WIDTH or *PW*

Specifies the page width of the OUTPUT file. The value specified is the right margin of the OUTPUT file. The value must be at least 64 greater than the value of the INITIAL_INDENT_COLUMN parameter. The default page width is 110.

INITIAL_INDENT_COLUMN or *IIC*

Specifies the starting column of the first line written to the OUTPUT file. This parameter ensures formatting of a section of commands not normally formatted (such as those appearing as input to COLLECT_TEXT). These commands are extracted from a procedure, formatted, and replaced in the formatted procedure. The default starting column is 1.

KEY_CHARACTER or *KC*

Specifies the character that causes the entire input line to be written to the output line without any format processing. This character must appear in column 1 of an input line. The default key character is the asterisk (*).

UTILITY_DEFINITION_FILE or *UDF*

Specifies the name of the file that lists the commands that initiate and terminate command utilities. If you create your own utilities, you need to create a file that lists all the command utilities used in the procedure.

If you omit this parameter, only system-defined utility names and terminators are included in the formatted output.

PROCESS_COLLECT_TEXT or *PCT*

Boolean value specifying whether lines within the COLLECT_TEXT command and other similar commands should be formatted. The default value is FALSE (lines should not be formatted).

Remarks

- If the output file named in the FORMAT_SCL_PROC command was not previously opened and any of its FILE_CONTENTS, FILE_PROCESSOR, and FILE_STRUCTURE attributes are set to UNKNOWN, the UNKNOWN value is changed as follows:
  - For the FILE_CONTENTS attribute, UNKNOWN is changed to LEGIBLE.
  - For the FILE_PROCESSOR attribute, UNKNOWN is changed to SCL.
  - For the FILE_STRUCTURE attribute, UNKNOWN is changed to DATA.
- For more information, see the NOS/VE System Usage manual.

FORTRAN Command

Purpose Calls and executes the FORTRAN compiler.

Format

FORTRAN or
FTN
INPUT=list of file
BINARY_OBJECT=file
LIST=file
COMPILATION_DIRECTIVES=boolean
DEBUG_AIDS=list of keyword
DEFAULT_COLLATION=keyword
ERROR=file
ERROR_LEVEL=keyword
EXPRESSION_EVALUATION=list of keyword
FORCED_SAVE=boolean
INPUT_SOURCE_MAP=list of file
LIST_OPTIONS=list of keyword
MACHINE_DEPENDENT=keyword
ONE_TRIP_DO=boolean
OPTIMIZATION=keyword
OPTIMIZATION_OPTIONS=list of keyword
RUNTIME_CHECKS=list of keyword

SEQUENCED_LINES=boolean
STANDARDS_DIAGNOSTICS=keyword
TERMINATION_ERROR_LEVEL=keyword
TARGET_MAINFRAME=keyword
STATUS=status variable

Parameters

INPUT or *I*

Specifies the name of the file(s) containing the input source code. Each file must contain a full program unit and not parts of a program unit. If more than one file is specified, the list of files is enclosed in parentheses.

Omission (or $INPUT) sets the specified file to $INPUT.

BINARY_OBJECT or *BINARY* or *B* or *BO*

Specifies the file to receive the binary object code produced by the compiler. The binary object code will be generated into a single file even if you specified a list of file references for the INPUT parameter.

Omission causes $LOCAL.LGO to be used.

LIST or *L*

Specifies the file to receive the compiler output listing. The format of the compiler output listing will be as though you had combined the input files into a single file even if you specified a list of file references for the INPUT parameter.

Omission causes $LIST to be used for batch jobs and $NULL to be used for interactive jobs.

COMPILATION_DIRECTIVES or *CD*

Determines whether C$ directives within the source program are recognized. If ON is selected, the C$ directives are processed. OFF causes the C$ directives not to be processed.

Omission causes this parameter to be set to ON.

DEBUG_AIDS or *DA*

Selects debugging options.

NONE

No debugging options are selected (default).

ALL

Selects all debugging options (DT and PC).

DT

Generates line number and symbol tables for use by the Debug utility.

PC

Generates argument checking information as part of the object code.

DEFAULT_COLLATION or *DC*

Specifies the weight table to be used for evaluating character expressions and by the CHAR and ICHAR functions. USER (U) allows a user-specified weight table (DISPLAY) to be used.

Omission, or FIXED (F), causes the fixed weight table (ASCII) to be used.

ERROR or *E*

Specifies the name of the file to receive compiler-generated error messages. Omission causes the error messages to be written to the file $ERRORS.

ERROR_LEVEL or *EL*

Determines the minimum severity level for which errors are to be listed. All errors of severity greater than or equal to the specified level will be listed on the error and list files. Options are:

TRIVIAL (T) or INFORMATIONAL (I)
WARNING (W)
FATAL (F)
CATASTROPHIC (C)

Omission causes WARNING to be used.

EXPRESSION_EVALUATION or *EE*

Controls the way the compiler evaluates expressions. Options are:

CANONICAL (C)

Causes expressions to be evaluated according to precedence rules. Refer to the FORTRAN usage manual for detailed information about precedence rules.

MAINTAIN EXCEPTIONS (ME)

Inhibits optimizations that eliminate instructions that might cause runtime errors.

MAINTAIN PRECISION (MP)

Inhibits optimizations that change a floating-point operation to a mathematically equivalent form.

OVERLAPPING_STRINGS_MOVES (OSM)

Guarantees valid character assignment in character assignment statements of the form v=exp where the character positions being defined in v are referenced in exp.

REFERENCE (R)

Causes intrinsic functions to be called by reference rather than by value and results in the generation of descriptive error messages (of execution errors) by internal FORTRAN routines. If this option is not selected, the operating system produces error messages which generally provide less information.

Multiple options are specified in the form (op, ..., op). Omission, or NONE, causes no options to be selected.

FORCED_SAVE or *FS*

Specifies whether or not the values of variables and arrays in subprograms are to be retained after execution of a RETURN or END statement.

ON causes variable and array values to be saved. This is equivalent to specifying a SAVE statement in every subprogram compiled.

Omission, or OFF, causes variable and array values not to be saved.

INPUT_SOURCE_MAP or *ISM*

Specifies the file containing the source map that was generated by the OUTPUT_SOURCE_MAP option on the SCU EXPAND_DECK command. Omission causes $NULL to be used.

LIST_OPTIONS or *LO*

Specifies the information that is to appear on the compiler output listing. Options are:

A

A listing of the attributes of each symbolic name used or defined within the program is produced.

R

A cross reference listing. The listing shows the locations of the definition and use of each symbolic name in the program.

M

A symbol attribute list (same as A option), DO loop map and common block map are produced.

S

A listing of the program source statements is written to the output file.

SA

Same as the S option, except that lines turned off by C$ LIST directives are listed.

O

A listing of the generated object code is provided. that

NONE

No output listing is produced.

Multiple options are specified in the form (op, ..., op). Omission causes the S option to be used.

MACHINE_DEPENDENT or *MD*

Specifies whether the use of machine dependent capabilities within the program are to be diagnosed and how severely.

NONE

Machine dependent usages are not diagnosed.

TRIVIAL (T) or INFORMATIONAL (I)

Machine dependent usages are diagnosed as trivial errors.

WARNING (W)

Machine dependent usages are diagnosed as warning errors.

FATAL (F)

Machine dependent usages are diagnosed as fatal errors, which result in a nonexecutable program.

Omission causes NONE to be used.

ONE_TRIP_DO or *OTD*

Determines the manner in which DO loops are to be optimized by the compiler. The trip count is the number of times a DO loop is executed.

ON informs the compiler that the minimum trip count for DO loops is one.

OFF informs the compiler that the minimum trip count for DO loops is zero.

Omission selects the OFF option.

OPTIMIZATION or *OPT* or *OPTIMIZATION_LEVEL* or *OL*

Selects the level of optimization performed by the compiler. Options are HIGH, LOW or DEBUG. DEBUG produces stylized object code for debugging. LOW produces optimized code for production runs.

Omission selects the LOW option.

OPTIMIZATION_OPTIONS or *OO*

Specifies instruction scheduling. This parameter is only significant when the OPTIMIZATION_LEVEL parameter specifies HIGH. Options are NONE and INSTRUCTION SCHEDULING. INSTRUCTION SCHEDULING allows for improved execution on the Model 990 regardless of the machine on which compilation occurs. NONE indicates

that instruction scheduling is determined by the values of the OPTIMIZATION_LEVEL and TARGET_MAINFRAME parameters as follows:

If TARGET_MAINFRAME specifies C180CM or C180V and the compilation machine is a model 990, instruction scheduling occurs. If TARGET_MAINFRAME specifies C180CM and the compilation machine is not a model 990, instruction scheduling also occurs. All other combinations of the TARGET_MAINFRAME and compilation machine do not produce instruction scheduling.

Omission selects NONE.

RUNTIME_CHECKS or *RC*

Specifies the checking done during execution.

R

Selects runtime range checking for character substring expressions.

S

Selects runtime range checking for subscript expressions.

NONE

Causes no options to be selected (default).

ALL

Selects both the R and S options.

SEQUENCED_LINES or *SL*

Specifies the sequencing format of the input source program. Specify ON if the source program is in sequenced format or OFF if the source program is in nonsequenced format.

Omission selects the OFF option.

STANDARDS_DIAGNOSTICS or *SD*

Specifies whether or not non-ANSI source statements are to be diagnosed and, if so, how severely.

NONE

Nonstandard usages are not diagnosed.

TRIVIAL (T) or INFORMATIONAL (I)

Nonstandard usages are diagnosed as trivial errors.

WARNING (W)

Nonstandard usages are diagnosed as warning errors.

FATAL (F)

Nonstandard usages are diagnosed as fatal errors.

Omission causes NONE to be used.

TERMINATION_ERROR_LEVEL or *TEL*

Specifies the minimum error severity level for which the compiler is to return abnormal status. Options are:

TRIVIAL (T) or INFORMATIONAL (I)
WARNING (W)
FATAL (F)
CATASTROPHIC (C)

Abnormal status is returned for all errors having severity equal or greater than the specified level. Omission selects the FATAL option.

TARGET_MAINFRAME or *TM*

The TARGET_MAINFRAME(TM) parameter specifies the kind of mainframe for which the object code is generated. This parameter is only significant when the OPTIMIZATION_LEVEL parameter specifies HIGH. Options are:

Omitted

Same as TARGET_MAINFRAME=C180_CURRENT_MAINFRAME.

TARGET_MAINFRAME=C180_VECTOR
(TM=C180V)

The object code is generated for use on the model 990 of the CYBER 180. The model 990 has vector-processing capabilities; object code produced by this option also executes on any CYBER 180 model, however, it performs optimally on a model 990.

TARGET_MAINFRAME=C180_MODEL_INDEPENDENT (TM=180MI)

The object code is generated for use on any model of the CYBER 180.

TARGET_MAINFRAME=C180_CURRENT_MAINFRAME (TM=C180CM)

The object code is generated for use on the machine on which compilation occurs.

Remarks

- The parameters on the FORTRAN command can be specified either by name or positionally (parameter name omitted), and must be separated by a comma or one or more blanks.
- When you specify a parameter positionally, you must indicate the position of any omitted parameters that precede the specified parameter by commas.
- If you omit any parameter from the FORTRAN command, a default is automatically provided.
- For more information, see the FORTRAN Language Definition manual.

Examples

The following commands specify three parameters and select the default values for all other parameters.

```
/fortran input=afile binary object=bfile ..
../error level=fatal
```

```
/ftn i=afile b=bfile el=fatal
```

Options chosen:

INPUT=AFILE, I=AFILE

Source statements are read from file AFILE.

BINARY OBJECT=BFILE, B=BFILE

Object code is written to file BFILE.

ERROR LEVEL=FATAL, EL=FATAL

Only fatal and catastrophic errors are written to the error and list files.

The following commands are equivalent, both select default values for all parameters.

```
FORTRAN or FTN
```

GENERATE_COMMAND_TABLE Command

Purpose Generates command and function tables for a command environment.

Format

GENERATE_COMMAND_TABLE or
GENCT
INPUT=file
OUTPUT=file
PAGE_WIDTH=integer
STATUS=status variable

Parameters

INPUT or I

Specifies the file which contains the command table declaration. This parameter is required.

OUTPUT or O

Specifies the file to which the CYBIL declarations that represent the command table are to be written. This parameter is required.

PAGE_WIDTH or *PW*

Specifies the desired page width of the output file (31..110)

Omission causes the page width attribute of the output file to be used. If the attribute value specified is outside of the range 31 to 110, the nearest limit will be the value used. If that attribute is not specified, a page width of 110 is used.

Remarks For more information, see the CYBIL System Interface manual.

GENERATE_MESSAGE_TEMPLATE Command

Purpose Generates message templates.

Format

GENERATE_MESSAGE_TEMPLATE or
GENERATE_MESSAGE_TEMPLATES or
GENMT
INPUT=file
OUTPUT=file
ERROR=file
PRODUCT_IDENTIFIER=name
STATUS=status variable

Parameters

INPUT or **I**

Specifies the file containing the CYBIL definitions for the condition codes, their severity levels and their message templates. This parameter is required.

OUTPUT or **O**

Specifies the file to receive the message module creation commands. This file must be included within the CREATE_OBJECT_LIBRARY utility to actually create the message module(s). This parameter is required.

ERROR or *E*

Specifies the file to which information about errors encountered in the input is written. Omission causes $ERRORS to be used.

PRODUCT_IDENTIFIER or *IDENTIFIER* or *PI*

Reserved.

Remarks For more information, see the CYBIL System Interface manual.

GENERATE_PDT Command

Purpose Interprets a PDT declaration and generates CYBIL statements that declare and initialize a PDT.

Format

```
GENERATE_PDT or
GENPDT
    INPUT=file
    OUTPUT=file
    PAGE_WIDTH=integer
    STATUS=status variable
```

Parameters

INPUT or I

Specifies the file containing the PDT declaration. Blank lines and continuation lines are allowed anywhere in the input file, but only one PDT declaration is allowed. This parameter is required.

OUTPUT or O

Specifies the file which is to receive the output from the GENPDT command. All lines from the input file are echoed to the output file in the form of block comments. This parameter is required.

PAGE_WIDTH or PW

Specifies the desired page width of the output file. The range is 31 through 110. If the value specified is outside the range, the nearest limit will be used.

Omission causes the page width attribute of the output file to be used. If that attribute is not specified, a value of 110 is used.

Remarks For more information, see the CYBIL System Interface manual.

GENERATE_SCU_EDIT_COMMANDS Command

Purpose Compares a deck to text on a source file and produces a file of editing commands and text. If the deck is subsequently edited using the file of commands and text as input, the text of the edited deck would match that on the source file.

Format

GENERATE_SCU_EDIT_COMMANDS or
GENSEC
DECK=name or **keyword**
SOURCE=file
EDIT_COMMANDS=file
BASE=file
TERMINATING_DELIMITER=string
LEADING_SPACES_SIGNIFICANT=boolean
STATUS=status variable

Parameters

DECK or **D**

Deck to which the editor subcommands apply. You can specify a name or the keyword ALL. ALL specifies that the source file is to include the *DECK directives. Some libraries may have a key character other than *. This parameter is required.

SOURCE or **S**

File containing a modified version of the deck text. You can specify a file position as part of the file name. This parameter is required.

EDIT_COMMANDS or **EC**

File to which the editor commands and text are written. This parameter is required.

BASE or *B*

Source library file on which the deck resides. If BASE is omitted, SOURCE_LIBRARY is used.

TERMINATING_DELIMITER or *TD*

Characters that mark the end of inserted text in the editor commands file. If TERMINATING_DELIMITER is omitted, ///END\\\ is used.

LEADING_SPACES_SIGNIFICANT or *LSS*

Indicates whether the comparison should consider leading spaces significant. Options are:

TRUE

Leading spaces are significant.

FALSE

Leading spaces are not significant.

If LEADING_SPACES_SIGNIFICANT is omitted, TRUE is used.

Remarks

- The GENERATE_SCU_EDIT_COMMANDS command is a NOS/VE command. Although you can enter the command during an SCU session, it has no effect on the working library of the session.
- The source file text must not contain line identifiers. Also, the source file must not contain DECK directives unless DECK=ALL is specified.
- If it does not matter how many spaces precede the text in a line, specify LEADING_SPACES_SIGNIFICANT=FALSE, so that the command does not generate editing commands whose only function is to change the number of leading spaces.
- For more information, see the NOS/VE Source Code Management manual.

Examples

The following command compares the text on file NEW_DECK4 with the text in deck DECK4 on library OLDPL. It then writes a sequence of editing commands and text on permanent file DECK4_EDIT that, if executed, would change the deck text to match the text on file NEW_DECK4.

```
/generate_scu_edit_commands deck=deck4 ..
../source=$user.new_deck4  edit_commands=..
../$user.deck4_edit base=$user.oldpl
```

If you specify the parameter EDIT_COMMANDS=$USER.DECK4_EDIT on the GENSEC command, the following example gives the generated commands to the EDIT_FILE utility, and the utility makes a modification for the new version of your deck.

```
sc/edit_deck modification=new_mod deck=deck4 ..
sc../input=$user.deck4_edit
```

GET_FILE Command

Purpose Copies a file from NOS or NOS/BE to NOS/VE.

Format

GET_FILE or
GETF
TO=file
FROM=name
DATA_CONVERSION=keyword
USER=name
PASSWORD=list of name
CYCLE=integer
STATUS=status variable

Parameters

TO or **T**

Identifies the NOS/VE file to which the NOS or NOS/BE file is copied and, optionally, specifies how the file is to be positioned prior to use. This parameter is required.

FROM or *F*

Specifies the name of the NOS or NOS/BE file to be copied.

In NOS this file name can be up to seven characters in length as defined in the NOS file system.

In NOS/BE only the filename part of the file reference is used to access the NOS/BE file.

In either system omission of this parameter causes the file name component of the TO parameter to be used.

DATA_CONVERSION or *DC*

Specifies the type of conversion to be done during the file copy. The possible keywords are:

B60

Each 60-bit NOS or NOS/BE word is placed in the rightmost bits of each 64-bit NOS/VE word. The leftmost 4-bits of each NOS/VE word are set to 0 (zero).

B56

The rightmost 56 bits of each NOS or NOS/BE word are packed into contiguous NOS/VE bits. The leftmost 4 bits of each NOS or NOS/BE word are ignored. This

specification is useful for retrieving a file that was created on NOS/VE and then transferred to NOS/BE with the REPLACE_FILE command which also specified B56 as the DATA_CONVERSION parameter value.

A6

The NOS or NOS/BE file contains character data in the 6/12 display code format (ASCII 128-character set). Each character is converted to an ASCII character (7-bit code right-justified in an 8-bit byte).

A8

The NOS or NOS/BE file contains character data in the 12-bit ASCII code format (ASCII 128-character set). Each character is converted to an ASCII character (7-bit code right-justified in an 8-bit byte).

D63

The NOS or NOS/BE file contains character data in the 6-bit display code format (63 character subset of the ASCII 128-character set). Each character is converted to an ASCII character (7-bit code right-justified in an 8-bit byte).

D64

The NOS or NOS/BE file contains character data in the 6-bit display code format (64-character subset of the ASCII 128-character set). Each character is converted to an ASCII character (7-bit code right-justified in an 8-bit byte).

A63, B32, and B64 values are not supported. Omission of this parameter causes A6 to be used.

USER or *U* or *ID*

In NOS, USER specifies the NOS user name of the owner of the file. This parameter is necessary only if the file is registered in a catalog belonging to a user whose identification is different than your NOS identification (specified on a prior CHANGE_LINK_ATTRIBUTE command).

In NOS/BE, USER specifies the file id of the NOS/BE file. This parameter is not required if the NOS/BE file id is the same as the current NOS/VE user name or the user name specified in the last CHANGE_LINK_ATTRIBUTE command.

PASSWORD or *PASSWORDS* or *PW*

In NOS, PASSWORD specifies the NOS file password needed to gain access to the file. It is required only when the file does not belong to you.

In NOS/BE, PASSWORD specifies a list of up to two passwords that may be supplied to control file access (NOS/BE turnkey and read permissions are the only meaningful passwords for a GET_FILE command). The PASSWORD parameter is required only if turnkey and/or read permissions are required to access the file.

CYCLE or *CY* or *C*

The CYCLE parameter applies only to NOS/BE. This parameter specifies the NOS/BE file cycle number and is required only if the file being accessed is not the highest cycle cataloged.

Remarks

- A SET_FILE_ATTRIBUTE command may precede the GET_FILE command to establish attribute values that are to be preserved with the file. A subsequent REPLACE_FILE command does not preserve file attribute values.
- A NOS or NOS/BE permanent file that resides on online storage can be copied.
- A CHANGE_LINK_ATTRIBUTE command issued prior to the GET_FILE identifies the accounting and user identification information needed to access the file.
- If the data conversion is D64, A6, or A8, then the NOS or NOS/BE file must be formatted with zero-byte terminated (Z-type) records.
- Any embedded end-of-record marks (EORs) in the NOS or NOS/BE file are eliminated during transfer to the NOS/VE file.

- For more information, see the NOS/VE System Usage manual.

Examples NOS

The first example retrieves file A from the NOS system.

```
/get_file a
```

The next example retrieves a copy of a NOS file named PROLOG and stores the file in the master catalog with the NOS/VE file name PROLOG.

```
/get_file to=$user.prolog
```

NOS/BE

The first example accesses the NOS/BE file DATAFIL with the following file attributes:

```
file id      JLC
passwords    PERM1 and PERM2
cycle        32

/get_file to=datafil_copy from=datafil user=jlc ..
../password=(perm1 perm2) cycle=32
```

The next example retrieves a file A from the NOS/BE system with a file id equal to DWM for a NOS/VE user named DWM.

```
/get_file a
```

GET_MULTI_RECORD_FILE Command

Remarks Reserved for site personnel, Control Data, or future use.

HELP Command

Purpose Requests a description of a system message or displays the text of an online manual.

Format

HELP or
H

SUBJECT=subject
MANUAL=file
LIST=file
STATUS=status variable
CLV$PREVIOUS_STATUS=status

Parameters

SUBJECT or *S*

Specifies the index topic for locating information. Enclose the topic in apostrophes if the topic has spaces in it.

MANUAL or *M*

Specifies the online manual to be displayed. Omission causes the SCL Quick Reference manual to be displayed. If you are in a utility that has an online manual associated with it, the utility's online manual is used by default.

LIST or *L*

Specifies the file to receive listable output produced by the EXCERPT directive or the COPY key. Omission causes file MANUAL_PAGES to be used.

CLV$PREVIOUS_STATUS

Determines whether or not the previous status was normal.

Remarks

- Specifying HELP without parameters causes the display of message information if the last command you executed was not normal. Otherwise, it causes the display of the main menu of the SCL Quick Reference manual.
- With parameters, HELP can be used to specify a particular online manual or a particular subject.
- If you are within a utility, HELP causes the display of the online manual for your utility or the utility's subcommand list menu in the SCL Quick Reference manual.
- For more information, see the NOS/VE System Usage manual.

Examples The following example calls the online FORTRAN manual:

```
/help m=fortran
```

The following example displays the description of the $QUOTE function:

```
/help $quote
```

If you want the SUBJECT parameter to be evaluated, you must use the EXPLAIN command. For example, suppose the SCL string variable FTN were defined as follows:

```
/ftn='accept_line'
```

If you entered,

```
/help s=ftn
```

the system does not evaluate FTN as an SCL variable and therefore provides a description of the FTN command. If you entered,

```
/explain s=ftn m=scl
```

the system evaluates FTN as an SCL variable and therefore provides a description of the ACCEPT_LINE command.

IF
Control Statement

Purpose Provides for conditional execution of one or more statement lists.

Format

IF boolean expression THEN
    statement list
ELSEIF boolean expression THEN
    statement list
ELSE
    statement list
IFEND

Parameters

boolean expression

Specifies the condition that must be TRUE in order for the following statement list to be executed. This parameter is required.

statement list

Specifies the statements to be executed.

Remarks

- The boolean expressions on the IF and ELSEIF clauses are evaluated in turn until one with a TRUE value is found or until an ELSE clause is found. The statement list is executed next, and control passes to the statement following the IFEND clause.
- For more information, see the NOS/VE System Usage manual.

Examples

The following is an example of an IF statement within an SCL procedure. 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.

```
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
```

INCLUDE_COMMAND Command

Purpose

Causes the text of a specified string to logically replace the occurrence of the INCLUDE_COMMAND command, thus enabling you to construct a command through string manipulation and then have the command processed.

Format

INCLUDE_COMMAND or
INCC
COMMAND=string
ENABLE_ECHOING=boolean
STATUS=status variable

Parameters

COMMAND or **C**

Specifies the string to be interpreted and processed as a command. This parameter is required.

ENABLE_ECHOING or *ENABLE_ECHO* or *EE*

Specifies whether the command is echoed. If this parameter is set to FALSE, the command is not echoed back to the terminal. The default value is TRUE.

Remarks

- The command you construct has access to the same variables, conditions, and parameter substitutions as the INCLUDE_COMMAND command.
- This command accepts only one command (unlike the INCLUDE_LINE command).
- This command does not create an input control block (unlike the INCLUDE_LINE command). That is, the command included is not treated as if it were the statement list of an unlabeled BLOCK statement.
- For more information, see the NOS/VE System Usage manual.

Examples

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
```

INCLUDE_FILE Command

Purpose

Inserts a text file containing SCL statements into the command stream.

Format

INCLUDE_FILE or
INCF
FILE=file
PROMPT=string
UTILITY=name or *keyword*
STATUS=status variable

Parameters

FILE or F

Identifies the text file to be included. This parameter is required.

PROMPT or P

Specifies a prompt string to be used if the file is assigned to an interactive terminal. If you use the UTILITY parameter to associate a utility with the INCLUDE_FILE command, the utility's prompt is used.

If you omit this parameter and do not specify an associated utility, the prompt *incf* is used.

UTILITY or U

Specifies the name of the utility to be associated with the INCLUDE_FILE command. To avoid association with a utility, use the keyword NONE.

If you omit this parameter, NONE is assumed.

Remarks

- The inserted text logically replaces the INCLUDE_FILE command.
- You can use this command either on its own or with a command utility (to initiate command processing for the utility).
- The inserted statement list is treated as if it were the statement list of an unlabeled BLOCK statement. When you exit the included file, the block no longer exists. Consequently, if you call a command utility within the statement list, and exit from the included file, the utility is terminated even though no explicit termination command (such as QUIT) was encountered.
- For more information, see the NOS/VE System Usage manual.

Examples Suppose that file $LOCAL.CREOL_COMMANDS contains the following commands:

```
add_module l=$local.programs_lgo
replace_module l=$local.lgo
```

Entering the following CREATE_OBJECT_LIBRARY and INCLUDE_FILE commands produces the results described below:

```
/creol
COL/include_file f=$local.creol_commands
COL/
```

The commands on file $LOCAL.CREOL_COMMANDS are executed. The contents of the file $LOCAL.PROGRAMS_LGO are added to the module list, and are then replaced by modules of the same name (if present) from the file $LOCAL.LGO.

INCLUDE_LINE Command

Purpose Inserts a string containing SCL statements into the command stream.

Format

INCLUDE_LINE or
INCL
STATEMENT_LIST=string
ENABLE_ECHOING=boolean
UTILITY=name or *keyword*
STATUS=status variable

Parameters

STATEMENT_LIST or **SL**

Specifies the string to be processed as a statement list. This parameter is required.

ENABLE_ECHOING or *ENABLE_ECHO* or *EE*

Specifies whether commands can be echoed (YES) or not (NO).

If you omit this parameter, YES is used.

UTILITY or *U*

Specifies the name of the command utility to be associated with the INCLUDE_LINE command. To avoid association with a utility, use the keyword NONE.

If you omit this parameter, NONE is used.

Remarks

- The specified statement list logically replaces the INCLUDE_LINE command.
- You can use this command either on its own or with a command utility (to initiate command processing for the utility).
- The inserted statement list is treated as if it were the statement list of an unlabeled BLOCK statement. When you exit the included line, the block no longer exists. Consequently, if you call a command utility within the statement list, and exit from the included line, the utility is terminated even though no explicit termination command (such as QUIT) was encountered.
- For more information, see the NOS/VE System Usage manual.

Examples

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 command_list
 3:26 PM
March 28, 1987
```

INITIALIZE_TERMINAL Command

Purpose

This command sends character sequences to change the terminal settings according to the current mode of system interaction. For the INITIALIZE_TERMINAL command to change the terminal settings, the terminal definition must contain at least two APPLICATION_STRING statements of the format:

```
APPLICATION_STRING NAME='LINE_INIT' OUT='text sent t-
o ..
 the terminal'

APPLICATION_STRING NAME='SCREEN_INIT' OUT='text sent-
 to ..
 the terminal'
```

where you supply the character string for the OUT parameter.

Format **INITIALIZE_TERMINAL** or
INIT
STATUS=status variable

Remarks For more information, see the NOS/VE Terminal Definition manual.

$INTEGER Function

Purpose Converts a string or boolean value to an integer.

Format **$INTEGER**
(any)

Parameters **any**

Specifies the string or boolean value you want converted to an integer. This parameter is required.

Remarks

- If you use this function to convert a string, you must be sure that the string contents conform to the rules for forming an integer constant.
- Boolean values are converted to 0 when the value is FALSE, and to 1 when the value is TRUE.
- For further information about functions, see the NOS/VE System Usage manual.

Examples

- The following examples show the creation of string variables and the use of the $INTEGER function to convert these strings to integers.

```
/create_variable sample_1 kind=string ..
../value='0ff(16)'
/display_value $integer(sample_1)
OFF(16)

/create_variable sample_2 kind=string ..
../value='123(10)'
/display_value $integer(sample_2)
123(10)
```

  SCL maintains the radix for each integer variable you create. If you want to know the current integer value of a variable, reference the variable by name; SCL returns the integer with the same radix you used when creating the variable.

- The following example converts a boolean value to an integer:

```
/display_value $integer(false)
0
```

$INTERACTION_STYLE Function

Purpose Returns a boolean value indicating whether the specified interaction style is in effect.

Format

$INTERACTION_STYLE
(keyword)

Parameters

keyword

Specifies the interaction style you want verified. This parameter is required. Use one of the following entries:

LINE (L)
SCREEN (S)

If the system returns a value of TRUE, the interaction style you specified is in effect. If the system returns a value of FALSE, the other interaction style is in effect.

Remarks For further information about functions, see the NOS/VE System Usage manual.

Examples The following example indicates that the current interaction style is screen mode:

```
/display_value $interaction_style(screen)
TRUE
```

JOB Command

Purpose Delimits a sequence of one or more SCL commands that are to be submitted to NOS/VE for batch processing.

Format

JOB
USER_JOB_NAME=name
CPU_TIME_LIMIT=integer or *keyword*
JOB_ABORT_DISPOSITION=keyword
JOB_CLASS=name
JOB_EXECUTION_RING=integer
JOB_QUALIFIER=list of name or *keyword*
JOB_RECOVERY_DISPOSITION=keyword
MAGNETIC_TAPE_LIMIT=integer or *keyword*
MAXIMUM_WORKING_SET=integer or *keyword*
OPERATOR_FAMILY=name
OPERATOR_USER=name
OUTPUT_DISPOSITION=file or *keyword*
SRU_LIMIT=integer or *keyword*
STATION=name or *keyword*
SUBSTITUTION_MARK=string or *keyword*
USER_INFORMATION=string
SYSTEM_JOB_NAME=string variable
STATUS=status variable

Parameters *USER_JOB_NAME* or *JOB_NAME* or *JN* or *UJN*

Specifies the user-supplied name by which the submitted job is to be known. Omission causes the user name of the submitting job to be used.

CPU_TIME_LIMIT or *CTL*

Specifies the maximum CPU time in seconds that will be allocated to the job. If the value specified is greater than the maximum CPU time limit allowed for the user by the site, the job will be terminated immediately. During job execution, if the job's accumulated job and monitor CPU time exceeds the value specified here, a job abort limit condition will occur and the job will be terminated.

If this parameter is specified, the job classes that the job can be a member of may be restricted. The job may only be a member in a job class that supports a CPU time limit greater than or equal to this parameter's value.

Specify one of the following on this parameter:

integer

Maximum CPU seconds allocated for the job.

SYSTEM_DEFAULT

The system default value for this attribute is to be used.

UNLIMITED

The maximum value allowed by the system is to be used.

If this parameter is omitted and an explicit job class name is specified on the JOB_CLASS parameter of this command, the CPU time limit is determined from the job class's CPU time limit or the user's validation CPU time limit, whichever is less. If the JOB_CLASS is AUTOMATIC, the CPU time limit for the job is determined by the system default value or the user's validation CPU time limit, whichever is less.

JOB_ABORT_DISPOSITION or *JAD*

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.

JOB_CLASS or *JC*

Specifies the name of the job class to be used for this job. If this parameter is omitted, the default job class of the login user is used. If that value does not exist, the system default is used.

The name AUTOMATIC (if the user is validated for job class AUTOMATIC) will automatically assign the job to a job class based on the job's attributes.

JOB_EXECUTION_RING or *JER*

Specifies the job's execution ring. Allowable values are from 4 through 13, but must be greater than or equal to the user's minimum ring validation. If this parameter is omitted, the user's nominal ring is used.

JOB_QUALIFIER or *JOB_QUALIFIERS* or *JQ*

Specifies from one to five site-defined names used to possibly limit a job to a specific job class or set of classes or mainframes.

Specify one of the following:

list of names

Up to five site defined job qualifiers.

NONE

No job qualifiers are to be used.

SYSTEM_DEFAULT

The system default value for this attribute is to be used.

If this parameter is omitted, the system default value is used.

JOB_RECOVERY_DISPOSITION or *JRD*

Specifies what should be done with the job by the active job recovery process if there is a system interrupt while the job is executing. If this parameter is omitted, then the system default is used. 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 continues normal execution. If the attempt fails, the action specified by the JOB_ABORT_DISPOSITION parameter is used.

RESTART

The job is automatically resubmitted so that it starts over from the beginning.

TERMINATE

The job is discarded.

MAGNETIC_TAPE_LIMIT or *MTL*

Specifies the maximum number of magnetic tape drives required simultaneously by the job.

If this parameter is specified, the job classes that the job can be a member of may be restricted. The job may only be a member in a job class that supports a magnetic tape limit greater than or equal to this parameter's value.

Specify one of the following on this parameter:

integer

Maximum number of tape units required by the job.

SYSTEM_DEFAULT

The system default value for this attribute is to be used.

UNLIMITED

The maximum value allowed by the system is to be used.

UNSPECIFIED

Assigns the value that is used when the parameter is omitted.

If this parameter is omitted, the system default is used.

MAXIMUM_WORKING_SET or *MAXWS*

Specifies the maximum working set in pages that the job requires. If this parameter is specified, the job classes that the job can be a member of may be restricted. The job may only be a member in a job class that supports a maximum working set size greater than or equal to this parameter's value.

Specify one of the following on this parameter:

integer

Maximum working set in pages needed for the job.

SYSTEM_DEFAULT

The system default value for this attribute is to be used.

UNLIMITED

The maximum value allowed by the system is to be used.

If this parameter is omitted, this values is determined by the job class specified on the JOB_CLASS parameter of this command. If the JOB_CLASS parameter is also not specified, the system default value is used.

OPERATOR_FAMILY or *OF*

Specifies the default private station or remote system operator family name attribute for output files generated by this job. 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 parameter is also used to establish the control family attribute of the output file. This parameter is not meaningful unless the OUTPUT_DESTINATION_USAGE job attribute is PRIVATE.

OPERATOR_USER or *OU*

Specifies the default private station or remote system operator user name attribute for output files generated by this job. This user name together with the OPERATOR_USER attribute identifies the private station operator or remote system operator who can print or receive the file. This parameter is also used to establish the CONTROL_USER attribute of the output file. This parameter is not meaningful unless the OUTPUT_DESTINATION_USAGE job attribute is PRIVATE.

OUTPUT_DISPOSITION or *ODI*

Specifies how the job's standard output is to be disposed. Allowable values are either a file name or one of several keywords. The following list describes the results of each of the allowable values. If this parameter is omitted, PRINTER is used.

file_name

Specification of a file name indicates that the standard output is to be copied to the specified permanent file at job end. Specification of a file that resides on a remote systems is not currently supported.

DISCARD_ALL_OUTPUT (DAO)

All output files generated by the job are discarded.

DISCARD_STANDARD_OUTPUT (DSO)

Standard output is to be discarded at job end.

PRINTER (P)

Any output generated by the job is printed at your default output station.

WAIT_QUEUE (WQ)

Any output files generated by the job are returned to the originating user's $WAIT_QUEUE subcatalog using the user's job name for the output file name or the user's file name for output generated by the PRINT_FILE command. If the $WAIT_QUEUE subcatalog does not exist at the time the output files are returned, it will be created for the user.

SRU_LIMIT or *SL*

Specifies the maximum system resource units (SRUs) that will be allocated to the job. If the value specified is greater than the maximum SRU limit allowed for the user by the site, the job will be terminated immediately. During job execution, if the job's accumulated SRUs exceed the value specified here, a job abort limit condition will occur and the job will be terminated.

If this parameter is specified, the job classes that the job can be a member of may be restricted. The job may only be a member in a job class that supports an SRU limit greater than or equal to this parameter's value.

Specify one of the following on this parameter:

integer

Maximum SRUs allocated for the job.

SYSTEM_DEFAULT

The system default value for this attribute is to be used.

UNLIMITED

The maximum value allowed by the system is to be used.

If this parameter is omitted and an explicit job class name is specified on the JOB_CLASS parameter of this command, the SRU limit is determined from the job class's SRU limit or the user's validation SRU limit, whichever is less. If the JOB_CLASS parameter is also not specified, the SRU limit for the job is determined by the system default value or the user's validation SRU limit, whichever is less.

STATION or *S*

Specifies the default I/O station name for output files generated by the job.

If the JOB_DESTINATION_USAGE attribute specifies PRIVATE, this parameter must specify the control facility name.

The keyword, AUTOMATIC, indicates that the system default is to be used.

SUBSTITUTION_MARK or *SM*

Specifies a one-character string used within a statement to delimit the text to be substituted. This character cannot be any character that is valid in an SCL name.

Corresponding pairs of substitution marks must appear on the same line.

If a second substitution mark is not found on the same line, the end-of-line is treated as the second mark.

If two consecutive substitution marks appear, they are replaced by a single substitution mark in the text.

Substitution marks cannot be used to construct the JOBEND statement.

USER_INFORMATION or *UI*

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.

If omitted, the user information string associated with the submitted job is assumed.

SYSTEM_JOB_NAME or *SJN*

Specifies a string variable to which the system supplied name of the job is returned. Omission causes this value to not be returned.

Remarks

- LOGIN and LOGOUT commands are generated by NOS/VE and added to the statements you supply.
- Your family name, user name, and password along with the specified job class and job name parameters are used to generate the LOGIN command. Validation of these values is done within the new batch job and not during the processing of the JOB command.
- This command is especially useful in interactive mode where you might submit several commands to execute in batch mode on a one-time basis.
- The commands (statement list) to be processed as a separate job are specified between the JOB and JOBEND statements.
- The SUBSTITUTION_MARK parameter enables a batch job to be constructed using variables and procedure parameters contained within the initiating job. If you want to pass parameters to a submitted job, these parameters must be substituted directly into the statements of the submitted job.
- The JOB command cannot be used to execute a job on a remote system. Use the SUBMIT_JOB command to submit jobs on remote systems.
- For more information, see the NOS/VE System Usage manual.

Examples The following example creates a batch job that compiles a CYBIL program, prints the compiler listing, and executes the object file:

```
/job job_name=compile
job/cybil i=$user.cybil_program l=list_file
job/print_file list_file
job/lgo
job/jobend
```

The following example illustrates the use of the JOB/JOBEND command using the SUBSTITUTION_MARK parameter:

```
proc compile_fortran (fortran_input:  file=$required)

   job substitution_mark='%'
     fortran i=%$string($value(fortran_input))% ..
       optimization_level=high b=$local.lgo
     $local.lgo
   jobend

procend compile_fortran
```

$JOB Function

Purpose Returns specified attributes of the current job.

Format

$JOB
(keyword)

Parameters

keyword

Specifies the job attribute(s) you want returned. Chapter 3, Function Attributes, lists and describes the keyword values you can supply and the corresponding function results. This parameter is required.

Remarks

- The kind of result returned depends on the attribute being tested. When a string value is returned by the function, all letters are converted to uppercase.
- For further information about functions, see the NOS/VE System Usage manual.

Examples The following example from an SCL procedure returns the mode of the current job. If the mode is interactive, a message is displayed.

```
IF $job(mode) = 'INTERACTIVE' THEN
    display_value 'Welcome to NOS/VE.'
IFEND
```

$JOB_DEFAULT Function

Purpose Returns the system default values for a job's attributes.

Format

$JOB_DEFAULT or
$JOB_DEFAULTS
(keyword1
keyword2)

Parameters

keyword1

Specifies the attribute you want returned.

Refer to appendix C for a listing of the job attribute names and the corresponding function results. This parameter is required.

keyword2

Specifies the job mode for which you want the system default information. Options are:

BATCH (B)

Returns information specific to batch jobs.

INTERACTIVE (I)

Returns information specific to interactive jobs.

If this parameter is omitted, the job mode of the issuing job is used.

Remarks For more information, see the NOS/VE System Usage manual.

$JOB_LIMIT Function

Purpose Returns information about specified job resource limits.

Format

```
$JOB_LIMIT
    (name,
    keyword)
```

Parameters

name

Specifies the job resource limit for which you want information returned. Use one of the following names:

CP_TIME

Specifies the CP time, in microseconds.

SRU

Specifies the system resource units that your job accumulates.

TASK

Specifies the task currently being executed within your job.

keyword

Specifies the form in which you want the information returned. This parameter is required. Use one of the following keywords:

ACTIVE

Indicates whether the specified limit is active for the job. If you specify this keyword, the $JOB_LIMIT function returns a boolean value.

ACCUMULATOR

Specifies the current accumulator value. If you specify this keyword, the $JOB_LIMIT function returns an integer value.

RESOURCE_LIMIT

Specifies the current resource limit value. If you specify this keyword, the $JOB_LIMIT function returns an integer value.

ABORT_LIMIT

Specifies the maximum allowed resource limit value, that is, the point at which your job will abort. If you specify this keyword, the $JOB_LIMIT function returns an integer value.

Remarks For further information about functions, see the NOS/VE System Usage manual.

Examples The following example changes a job's CP time resource limit so that it is 60 CP seconds over the current accumulator value:

```
/change_job_limit name=cp_time ..
../resource_limit=($job_limit(cp_time, ..
../accumulator)+60)
```

$JOB_OUTPUT Function

Purpose Returns attribute information about an output file.

Format

$JOB_OUTPUT
(name,
keyword)

Parameters

name

Specifies the output file whose attributes you want returned. Use either a system-supplied or user-supplied file name. A user-supplied name must be unique; otherwise the function will fail. This parameter is required.

keyword

Specifies the attribute you want returned.

Chapter 3, Function Attributes, lists and describes the keyword values you can supply and the corresponding function results. This parameter is required.

Remarks

- The type of result returned depends on the attribute being tested.
- For further information about functions, see the NOS/VE System Usage manual.

Examples The following example returns the file size of file LIST_FILE:

```
/print_file list_file du=private fc='xx'
/display_value $job_output(list_file,file_size)
98
```

$JOB_STATUS Function

Purpose Returns the status of a job.

Format

$JOB_STATUS
(name,
keyword)

Parameters

name

Specifies the name of the job for which you want the status returned. Either a system-supplied or user-supplied job name is valid. If you specify a user-supplied job name, it must be unique; otherwise the function will fail. This parameter is required.

keyword

Specifies the type of information you want returned.

Chapter 3, Function Attributes, lists and describes the keyword values you can supply and the corresponding function results. This parameter is required.

Remarks

- The user of the function must be the login user, the control user, a system operator, or the immediate parent job, that is, the job's submitter. (A login user is the user name under which the job is scheduled and run; for most jobs, the control user is the login user.)
- You can use the JOB_STATE attribute to test for the existence of a particular job.
- The type of result returned depends on the attribute being tested.
- For further information about functions, see the NOS/VE System Usage manual.

Examples The following example returns the status of a job using the user-supplied job name:

```
/display_value $job_status(sueo_24,state)
INITIATED
```

KERMIT Command

Purpose Begins a KERMIT file transfer session.

Format

KERMIT
TAKE=file
STATUS=status variable

Parameters *TAKE* or *T*

Reserved.

Remarks For more information, see the online KERMIT manual.

LINK_ADA Command

Purpose Links Ada code after it has been compiled and before it can be executed.

Format

LINK_ADA or
LINA
MAIN_PROGRAM=name
PROGRAM_LIBRARY=file
BINARY=file
LIST=file
RECOMPILATIONS=string
STATUS=status variable

Parameters **MAIN_PROGRAM** or **MP**

Specifies the compilation unit to be linked, that is, the name of the function or procedure to be executed. The function or procedure must have been compiled so that it is a library unit in the sublibrary specified by the PROGRAM_LIBRARY parameter. The compilation unit names can be listed using the SHOW command in a PLU session. This parameter is required.

PROGRAM_LIBRARY or *PL*

Specifies the file containing the sublibrary to be referenced by the linker. The default value is $USER.ADA_PROGRAM_LIBRARY.

BINARY or *B*

Specifies the file on which the executable code extracted from the user's program library is written, thus creating an object file acceptable to the CYBER 180 loader. If $NULL is specified, the Ada linker performs all the compilation order validation checks, but does not create an object file. The default value is $LOCAL.LGO.

LIST or *L*

Specifies the file where the linker writes the library units elaboration order list. The default value is $LIST.

RECOMPILATIONS or *R*

Specifies the name or names of any modules that need to be recompiled. This parameter must be omitted to produce a binary file.

Remarks

- The main program name for the Ada linker is a procedure name used in the source text.
- The default binary file name, $LOCAL.LGO, is also the default file name for the EXECUTE_TASK command.
- For more information, see the ADA for NOS/VE Usage manual.

Examples

The following link command produces a list of dependencies. A binary file is not produced:

```
/lina mp=your_procedure recompilations l=dependencies_list
```

The following link command produces a binary file. A list of dependencies is not produced:

```
/lina mp=your_procedure b=binary_file
```

NOTE

The main program for the LINK_ADA Command must be a parameterless procedure.

LINK_DM
IM/DM Command

Purpose Links the specified program with the DM run time library and the DM library routine.

Format

LINK_DM or
LIND or
LINKDM
BINARY_OBJECT=dm_file_descriptor
EXECUTABLE=dm_file_descriptor
PROGRAM_DESCRIPTOR=name
LIBRARY=list of dm_file_descriptor
DMRWLIB=dm_file_descriptor or *keyword*
DEBUG=keyword
OPTION_FILE=dm_file_descriptor or *keyword*
MAP=dm_file_descriptor or *keyword*
DMSHR=keyword
SLANG=keyword
OMIT_DEBUG_TABLE=keyword
STARTING_PROCEDURE=name
LOAD_MAP=dm_file_descriptor or *keyword*
LOAD_MAP_OPTIONS=list of keyword
ABORT_FILE=dm_file_descriptor
INCLUDE_BINARY_SECTION_MAPS=keyword
PRELINK=keyword
MAX_MODULES=integer
STATUS=status variable

Parameters

BINARY_OBJECT or *OBJECT* or *OBJ* or *OLB* or *B*

Name of the program module to be linked. If the file type is not entered on the program name, it defaults to the object library (program_OLB).

EXECUTABLE or *EXE* or *E*

File on which the command writes an object library containing the program description and the bound module.

If EXECUTABLE is omitted, file program_EXE in the current working catalog is used.

PROGRAM_DESCRIPTOR or *PD*

Name given to the program description. (The name used to execute the program.)

If PROGRAM_DESCRIPTOR is omitted, program_PD is used.

LIBRARY or *LIBRARIES* or *LIB* or *L*

Object library files searched to satisfy external references (1 through 20 files).

If LIBRARY is omitted, no additional libraries are added to the list.

DMRWLIB

Report writer object library file used by the link. The keyword YES specifies the file DMRW_OLB. The parameter is used only when linking DMFQM with DMRW.

If DMRWLIB is omitted, no linking with DMRW is done.

DEBUG

Indicates whether the program is linked for debugging.

YES

Program is not bound, debug tables are kept, and a minimal program descriptor is generated so that most program attributes are taken from job defaults.

NO

Program is not linked for debugging.

If DEBUG is omitted, NO is used.

OPTION_FILE or *OPT*

File specifying a set of object libraries to be searched before the object libraries specified by the LIBRARY parameter.

The keyword YES specifies that the file program_OPT in the current working catalog is used.

If OPT is omitted, YES is used.

MAP or *M*

File where the section map is written. The keyword YES specifies the file program_MAP.

If MAP is omitted, no section map is generated.

DMSHR or *DMSHARE*

Indicates whether the DM run time library routines should be bound into the executable library.

YES

DM run time routines are included.

NO

DM run time routines are not included.

Only DM run time libraries with ring attributes of (11,11,11) are used.

If DMSHARE is omitted, YES is used unless PRELINK=YES is specified in which case NO is used.

SLANG

Indicates whether the program being linked is a SLANG program.

YES

The program is a SLANG program so the CYBIL SLANG main routine is included.

NO

The program is not a SLANG program.

If SLANG is omitted, NO is used.

OMIT_DEBUG_TABLE or *OMIT_DEBUG_TABLES* or *ODT*

Indicates whether Debug tables are removed by the link.

YES

Debug tables are omitted.

NO

Debug tables are kept.

If OMIT_DEBUG_TABLE is omitted, NO is used.

STARTING_PROCEDURE or *SP*

Name of the starting procedure for the program.

LOAD_MAP or *LM*

File on which the load map is written. The keyword YES specifies file program_LOAD_MAP in the current working catalog.

If LOAD_MAP is omitted, $NULL is used (a load map is not written).

LOAD_MAP_OPTIONS or *LOAD_MAP_OPTION* or *LMO*

Information included in the load map.

If LOAD_MAP_OPTIONS is omitted, no information is written to the load map.

ABORT_FILE or *AF*

File containing Debug commands to execute if the program aborts.

If ABORT_FILE is omitted, $NULL is used (no abort file is defined for the program).

INCLUDE_BINARY_SECTION_MAPS or *IBSM*

Reserved.

PRELINK or *PL*

Indicates whether the program is to be prelinked.

YES

Prelink the program.

NO

Do not prelink program.

Omission causes NO to be used.

MAX_MODULES or *MM*

Reserved.

Remarks For more information, see the DM Utilities Reference Manual for DM on CDC NOS/VE.

LINK_VIRTUAL_ENVIRONMENT Command

Remarks Reserved for site personnel, Control Data, or future use.

LISP Command

Purpose Executes the List Processing (LISP) interpreter.

Format

LISP
INPUT=file
OUTPUT=file
STATUS=status variable

Parameters

INPUT or *I*

Specifies the file reference for the file containing LISP statements to be evaluated. If you omit this parameter, the file $LOCAL.$INPUT is used and you are prompted for input at your terminal.

OUTPUT or *O*

Specifies the file reference for the file to receive output values or diagnostic messages. If you omit this parameter, the file $LOCAL.$OUTPUT (the terminal) is used.

Remarks For more information, see the LISP for NOS/VE Usage manual.

Examples The following command executes LISP using the terminal for both input and output.

```
LISP
```

LOGICAL_CONFIGURATION_UTILITY Command

Remarks Reserved for site personnel, Control Data, or future use.

LOGIN Command

Purpose Provides access to the NOS/VE batch services. Login parameters can be specified when you log in interactively through NAMVE/CDCNET.

Format

LOGIN
LOGIN_USER=name
PASSWORD=name
LOGIN_FAMILY=name
LOGIN_ACCOUNT=name
LOGIN_PROJECT=name
CPU_TIME_LIMIT=integer or *keyword*
JOB_ABORT_DISPOSITION=keyword
JOB_CLASS=name
JOB_EXECUTION_RING=integer
JOB_QUALIFIER=list of name or *keyword*
JOB_RECOVERY_DISPOSITION=keyword
MAGNETIC_TAPE_LIMIT=integer or *keyword*
MAXIMUM_WORKING_SET=integer or *keyword*
SRU_LIMIT=integer or *keyword*
USER_JOB_NAME=name

Parameters

LOGIN_USER or **USER** or **U** or **LU**

Specifies your user name. This is the user name assigned by your family administrator. This parameter is required.

PASSWORD or **PW**

Specifies your password. It is compared with the password registered with your validation information to verify your identity. This parameter is required.

LOGIN_FAMILY or *FAMILY_NAME* or *FN* or *LF*

Specifies the name of your family. If omitted, the default family name defined by your site administrator is used.

LOGIN_ACCOUNT or *ACCOUNT* or *A* or *LA*

Specifies the account to which your resource usage will be charged. If omitted, the default value established by your family administrator is verified.

LOGIN_PROJECT or *PROJECT* or *P* or *LP*

Specifies the project to which your resource usage will be charged. If omitted, the default value established by your family administrator is verified.

CPU_TIME_LIMIT or *CTL*

Specifies the maximum cpu time in seconds that will be allocated to the job. If the value specified is greater than the maximum cpu time limit allowed for the user by the site, the job will be terminated immediately. During job execution, if the job's accumulated job and monitor cpu time exceeds the value specified here, a job abort limit condition will occur and the job will be terminated.

If this parameter is specified, the job classes that the job can be a member of may be restricted. The job may only be a member in a job class that supports a cpu time limit greater than or equal to this parameter's value.

Specify one of the following on this parameter:

integer

Maximum cpu seconds allocated for the job.

SYSTEM_DEFAULT

The system default value for this attribute is to be used.

UNLIMITED

The maximum value allowed by the system is to be used.

If this parameter is omitted and an explicit job class name is specified on the JOB_CLASS parameter of this command, the cpu time limit is determined from the job class' cpu time limit or the user's validation cpu time limit, whichever is less. If the JOB_CLASS is AUTOMATIC, the cpu time limit for the job is determined by the system default value or the user's validation cpu time limit, whichever is less.

JOB_ABORT_DISPOSITION or *JAD*

Specifies what should be done with the job if it aborts because of system failure. If neither the SUBMIT_JOB command nor this command specifies this parameter, then the system default is used. The keywords are:

RESTART

Job is automatically resubmitted so that it starts over from the beginning.

TERMINATE

Job is discarded.

JOB_CLASS or *JC*

Specifies the name of the class under which the job is to run. The login will not be successful if the requesting user is not validated to execute a job of the specified class or if the current status of the system is such that it cannot accept more jobs of this class.

The name AUTOMATIC (if the user is validated to use the job class AUTOMATIC) will automatically assign the job to a job class based on the job's attributes.

JOB_EXECUTION_RING or *JER*

Specifies the execution ring of the job. Enter a value between 4 and 13 that is greater than or equal to your minimum ring. If omitted, your nominal ring is used.

JOB_QUALIFIER or *JOB_QUALIFIERS* or *JQ*

Specifies from one to five site-defined names used to possibly limit a job to a specific job class or set of classes or mainframes.

Specify one of the following:

list of names

Up to five site defined job qualifiers.

NONE

No job qualifiers are to be used.

SYSTEM_DEFAULT

The system default value for this attribute is to be used.

If this parameter is omitted, the system default value is used.

JOB_RECOVERY_DISPOSITION or *JRD*

Specifies what the active recovery process should do with the job if there is a system interrupt while the job is executing. If neither the SUBMIT_JOB command nor this command specifies this parameter, then the system default is used. 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 continues normal execution. If the attempt fails, the value specified on the JOB_ABORT_DISPOSITION parameter is used.

RESTART

Job is automatically resubmitted so that it starts over from the beginning.

TERMINATE

Job is discarded.

MAGNETIC_TAPE_LIMIT or *MTL*

Specifies the maximum number of magnetic tape drives required simultaneously by the job.

If this parameter is specified, the job classes that the job can be a member of may be restricted. The job may only be a member in a job class that supports a magnetic tape limit greater than or equal to this parameter's value.

Specify one of the following on this parameter:

integer

Maximum number of tape units required by the job.

SYSTEM_DEFAULT

The system default value for this attribute is to be used.

UNLIMITED

The maximum value allowed by the system is to be used.

UNSPECIFIED

Assigns the value that is used when the parameter is omitted.

If this parameter is omitted, the system default is used.

MAXIMUM_WORKING_SET or *MAXWS*

Specifies the maximum working set in pages that the job requires. If this parameter is specified, the job classes that the job can be a member of may be restricted. The job may only be a member in a job class that supports a maximum working set size greater than or equal to this parameter's value.

Specify one of the following on this parameter:

integer

Maximum working set in pages needed for the job.

SYSTEM_DEFAULT

The system default value for this attribute is to be used.

UNLIMITED

The maximum value allowed by the system is to be used.

If this parameter is omitted, this values is determined by the job class specified on the JOB_CLASS parameter of this command. If the JOB_CLASS parameter is also not specified, the system default value is used.

SRU_LIMIT or *SL*

Specifies the maximum system resource units (srus) that will be allocated to the job. If the value specified is greater than the maximum sru limit allowed for the user by the site, the job will be terminated immediately. During job execution, if the job's accumulated srus exceed the value specified here, a job abort limit condition will occur and the job will be terminated.

If this parameter is specified, the job classes that the job can be a member of may be restricted. The job may only be a member in a job class that supports an sru limit greater than or equal to this parameter's value.

If this parameter is omitted and an explicit job class name is specified on the JOB_CLASS parameter of this command, the sru limit is determined from the job class' sru limit or the user's validation sru limit, whichever is less. If the JOB_CLASS is AUTOMATIC, the sru limit for the job is determined by the system default value or the user's validation sru limit, whichever is less.

USER_JOB_NAME or *JOB_NAME* or *JN* or *UJN*

Specifies a name for the new job being established. Omission causes your user name to be used.

Remarks

- LOGIN must be the first command in a batch job.
- For interactive access, the system prompts you for login information.
- LOGIN must be the first command in a set of directives sent from a remote system with the MANAGE_REMOTE_FILES command or the MFLINK command.
- Once validated, NOS/VE executes any existing prolog file associated with your identification.
- LOGIN is a control command. Thus, you can override LOGIN, but you cannot remove it from your command list.
- If LOGIN_FAMILY is a remote family and the SUBMIT_JOB command is not used to specify some other queue file application, the job is transferred to the remote system through QTF.
- For more information, see the NOS/VE System Usage manual.

Examples The following example represents a NOS/VE batch job.

```
login login_user=sdh password=pass456 ..
    login_family=nve
collect_text fortran_source
      program ctime
      character*8 time
      print*,'The current time is: ',time()
      stop
      end
**
fortran i=fortran_source
lgo
logout
```

The job creates a text file containing a FORTRAN program that displays the current time, calls the FORTRAN compiler to compile the program, executes the program, and logs out.

LOGOUT Command

Purpose Terminates a batch or interactive job.

Format **LOGOUT**

Parameters None.

Remarks

- NOS/VE automatically logs out a batch job if the end-of-information (EOI) is encountered on the initial command file.
- Any epilog file associated with your validation is executed.
- For an interactive job, logging out results in termination of the NOS/VE session. The terminal remains connected and the network allows you to select another application or disconnect the terminal.
- LOGOUT is a control command. Thus, you can override LOGOUT but you cannot remove it from your command list.
- For more information, see the NOS/VE System Usage manual.

Examples Following is an example of logging out of NOS/VE from an interactive session.

```
/logout
VEIAF   CONNECT TIME 00.01.52.
T13A26  - APPLICATION: bye
LOGGED OUT.

HOST DISCONNECTED   CONTROL CHARACTER=(ESC)
ENTER INPUT TO CONNECT TO HOST
```

LOOP
Control Statement

Purpose Causes unlimited repetition of a statement list.

Format

label: **LOOP**
statement list
LOOPEND *label*

Parameters

label

Specifies the name of the LOOP block. This label can be used by CYCLE and EXIT statements within the block.

statement list

Specifies the list of statements that reside in the block.

Remarks

- Exit from a LOOP statement is possible only via an explicit transfer of control (that is, via an EXIT statement).
- For more information, see the NOS/VE System Usage manual.

Examples The following example uses a LOOP statement to read lines from file INPUT. Each line entered is stored in string variable INPUT_STRING. When a null input line is entered (by performing a single carriage return, 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.

MAIL Command

Purpose Begins a MAIL/VE session.

Format

MAIL or
MAI
PROLOG=file
EPILOG=file
OUTPUT=file
STATUS=status variable

Parameters

PROLOG or *P*

File containing MAIL/VE subcommands, SCL commands, or SCL procedures executed before MAIL/VE accepts commands from the user. The default is $USER.MAIL.PROLOG. This file can be positioned.

EPILOG or *E*

File containing MAIL/VE subcommands, SCL commands, or SCL procedures executed when the QUIT subcommand is processed. All MAIL/VE subcommands except ACTIVATE_SCREEN are valid within the MAIL/VE epilog. The default is $USER.MAIL.EPILOG. This file can be positioned.

OUTPUT or *O*

Output file for all MAIL/VE subcommands. The default is $OUTPUT. This file can be positioned.

Remarks

- When you enter MAIL, the system verifies that you are registered by checking if your NOS/VE family name and user name are in the mail directory. If you are not authorized, command processing terminates with an error message.
- See the NOS/VE System Usage manual if you need information on file positioning.
- For more information, see the MAIL/VE online manual.

$MAINFRAME Function

Purpose Returns attributes of the hardware mainframe on which the job is currently executing.

Format

$MAINFRAME
(keyword)

Parameters

keyword

Name of the attribute you are querying. This parameter is required. Use one of the following names:

ACTIVE_PROCESSORS (ACTIVE_PROCESSOR or AP)

Returns an integer indicating the number of processors that are on.

IDENTIFIER (ID or I)

Returns a string that designates the mainframe.

PAGE_SIZE (PS)

Returns an integer that specifies the number of bytes in a memory page.

TOTAL_PROCESSORS (TOTAL_PROCESSOR or TP)

Returns an integer that indicates the total number of processors in the mainframe.

Remarks For further information about functions, see the NOS/VE System Usage manual.

Examples The following example returns a string that designates the mainframe:

```
/display_value $mainframe(identifier)
$SYSTEM_0860-0109
```

MANAGE_NETWORK_APPLICATIONS Command

Remarks Reserved for site personnel, Control Data, or future use.

MANAGE_REMOTE_FILE Command

Purpose Delimits a set of commands to be executed on the specified remote system.

Format

MANAGE_REMOTE_FILE or
MANAGE_REMOTE_FILES or
MANRF or
MFLINK
LOCATION=any
FILE=file
DATA_DECLARATION=keyword
UNTIL=string
SUBSTITUTION_MARK=string or *keyword*
STATUS=status variable

Parameters

LOCATION or **L**

Specifies the name of the remote location to be accessed. This is a name associated with a remote system, such as a family name or a logical identifier. (Location names are determined by your network application administrator.)

You cannot specify a variable name for this parameter. If you want to use a variable that has a name value, you can use the $NAME function instead.

This parameter is required.

FILE or **F**

Specifies the name of a file on the local NOS/VE system to be used as the input or output file during a file transfer. This parameter is required even when you are not performing a file transfer.

DATA_DECLARATION or *DD*

Specifies the data format of a file to be transferred.

If the remote location is another NOS/VE host, this parameter is ignored. The rules for copying NOS/VE files based on the local and remote file attributes apply. For a discussion of rules for copying NOS/VE files, see the NOS/VE System Usage manual.

If the remote location is a non-NOS/VE host, the following data descriptions are available. The meaning of each varies among the various remote host types. Refer to the Remote Host Facility Usage manual for system specific information

- C6

  Use this format when you transfer files to hosts using a six-bit code set. This format indicates the file contains character data from a character set with 64 or fewer character codes.

  The effect of this format is that that each machine sees the file in its native character set. Thus, if you transfer the file from NOS/VE to NOS, NOS/VE sends the file in ASCII and NOS receives it in display code. Transfers to other systems result in full ASCII transfers as if DD=C8 was used.

- C8

  This format has the following meanings depending upon which system the file is being transfered to:

  - NOS

    Transfer results in a NOS 8/12 ASCII file. Use the NOS FCOPY command to convert the file to NOS 6/12 format.

  - NOS/BE

    Same as for NOS.

- Any other ASCII system

  Transfer results in an ASCII file.

- IBM/MVS

  Transfer results in an EBCDIC file.

- UU

  Use this format to transfer binary files to remote systems. Object and source libraries should be transferred using this format. Files transferred to NOS or NOS/BE will be padded unless they end on a 120 bit boundry (this is because NOS and NOS/BE store their files in 60 bit format). Similarily, files transferred from NOS or NOS/BE to NOS/VE and that have a file length that is an odd multiple of 60 bits will be padded to the next full byte (8 bit) length.

UNTIL or *U*

Specifies the string indicating the end of commands in the list. The string must appear on a separate line. If this parameter is omitted, a string of two asterisks (**) is assumed.

SUBSTITUTION_MARK or *SM*

Specifies a character used to delimit text to be substituted within the command text following the MANAGE_REMOTE_FILES command. Values can be any character or the keyword NONE. NONE specifies that no substitution mark is to be used. If this parameter is omitted, NONE is assumed.

Remarks

- You must provide validation information required by the remote system. If this remote system is NOS/VE, the first command in the list of commands must be a LOGIN command. Alternately, you can issue a CREATE_REMOTE_VALIDATION command prior to using the MANAGE_REMOTE_FILES command.

- The names and parameters of commands accepted by each remote system type are described in the Remote Host Facility Usage manual.

  The MANAGE_REMOTE_FILES command passes the command text you supply to the remote system for execution. If the remote system is NOS/VE, the command text is a set of SCL commands to be executed as a batch job.

- You can include at most one remote command in the command text which causes an explicit file transfer. For remote NOS/VE systems, use the SEND_FILE or RECEIVE_FILE commands to explicitly transfer a file.

- For more information, see the NOS/VE System Usage manual.

$MAX_INTEGER Function

Purpose Returns the maximum positive integer allowed for a parameter.

Format **$MAX_INTEGER**

Parameters None.

Remarks

- The value returned is 9,223,372,036,854,775,807
- For further information about functions, see the NOS/VE System Usage manual.

Examples In the following example, the system is interrogated for the value of $MAX_INTEGER:

```
/display_value $max_integer
9223372036854775807
```

$MAX_NAME Function

Purpose Returns the maximum length of a name allowed for a parameter.

Format **$MAX_NAME**

Parameters None.

Remarks

- The value returned is 31.
- For further information about functions, see the NOS/VE System Usage manual.

Examples In the following example, the system is interrogated for the value of $MAX_NAME:

```
/display_value $max_name
31
```

$MAX_STRING Function

Purpose Returns the maximum length of a string allowed for a parameter.

Format **$MAX_STRING**

Parameters None.

Remarks

- The value returned is 256.
- For further information about functions, see the NOS/VE System Usage manual.

Examples In the following example, the system is interrogated for the value of $MAX_STRING:

```
/display_value $max_string
256
```

$MAX_VALUE_SETS Function

Purpose Returns the maximum number of value sets allowed for a parameter.

Format $MAX_VALUE_SETS

Parameters None.

Remarks

- The value returned is 2,147,483,647.
- For further information about functions, see the NOS/VE System Usage manual.

Examples In the following example, the system is interrogated for the value of $MAX_VALUE_SETS:

```
/display)_value $max_value_sets
2147483647
```

$MAX_VALUES Function

Purpose Returns the maximum number of values allowed per value set.

Format $MAX_VALUES

Parameters None.

Remarks

- The value returned is 2,147,483,647.
- For further information about functions, see the NOS/VE System Usage manual.

Examples In the following example, the system is interrogated for the value of $MAX_VALUES:

```
/display_value $max_values
2147483647
```

MEASURE_PROGRAM_EXECUTION Command

Purpose Starts a program measurement utility session.

Format

MEASURE_PROGRAM_EXECUTION or
MEAPE
STATUS=status variable

Remarks

- The session ends when you enter the subcommand QUIT. The descriptions of each program measurement subcommand follow this description.
- For more information, see the NOS/VE Object Code Management manual.

Examples The following utility session specifies the program as the modules on file LGO, executes the program, and saves the program profile on file MY_FILE.

```
/measure_program_execution
MPE/set_program_description target_text=lgo
MPE/execute_instrumented_task
MPE/display_program_profile output=my_file
MPE/quit
/
```

MERGE Command

Purpose Merges the sorted records from one or more files and writes the records in sorted order to a single output file.

Format

MERGE
FROM=list of file
TO=file
KEY=list of any
DIRECTIVES=list of file
LIST=file
LIST_OPTIONS=list of keyword
ERROR=file
ERROR_LEVEL=keyword
RESERVED_POSITION_9=boolean
ESTIMATED_NUMBER_RECORDS=range of integer
EXCEPTION_RECORDS_FILE=file
C170_COMPATIBLE=boolean

OMIT_DUPLICATES = boolean
OWNCODE_FIXED_LENGTH = integer
OWNCODE_MAXIMUM_RECORD_LENGTH = integer
OWNCODE_PROCEDURE_1 = name
OWNCODE_PROCEDURE_2 = name
OWNCODE_PROCEDURE_3 = name
OWNCODE_PROCEDURE_4 = name
OWNCODE_PROCEDURE_5 = name
RETAIN_ORIGINAL_ORDER = boolean
COLLATING_SEQUENCE_NAME = name
COLLATING_SEQUENCE_STEP = list of any
COLLATING_SEQUENCE_REMAINDER = boolean
COLLATING_SEQUENCE_ALTER = boolean
STATUS = status variable
SUM = list of any
ZERO_LENGTH_RECORDS = keyword
VERIFY_MERGE_INPUT_ORDER = boolean
LOAD_COLLATING_TABLE = list of name
RESULT_ARRAY = integer array

Parameters

FROM or *F*

List of 1 to 100 input files. (Merge input files must be presorted. For a merge with summing, input files must be presummed as well as presorted.)

If you omit the FROM parameter, the merge attempts to read input records from file OLD in the $LOCAL catalog.

TO or *T*

Output file.

If you omit the TO parameter and specify an owncode 3 procedure that does not return output to the merge, the merge does not require an output file. However, if you omit the TO parameter and specify an owncode 3 procedure that does return output to the merge, the merge writes the output to the default file NEW.

If you omit the TO parameter and the OWNCODE_PROCEDURE_3 parameter, the merge writes all output records to the default file NEW. If file NEW does not exist, the merge creates and uses file $LOCAL.NEW.

KEY or *K*

List of 1 through 106 key field definitions. A key field definition is a value set containing up to four values as follows:

(first..last,type,order) or

(first,length,type,order)

first

Position of the first byte of the key field within the record. (The leftmost byte in a record is numbered 1.)

last

Position of the last byte of the key field within the record.

length

Optional number of bytes in the key field. The default length is 1 byte.

type

Optional name of a numeric data format or collating sequence. The default key field type is ASCII.

The valid numeric data format names are:

BINARY
BINARY_BITS
INTEGER
INTEGER_BITS
NUMERIC_FS
NUMERIC_LO
NUMERIC_LS
NUMERIC_NS
NUMERIC_TO
NUMERIC_TS
PACKED
PACKED_NS
REAL

The predefined collating sequence names are: ASCII, ASCII6, COBOL6, DISPLAY, EBCDIC, EBCDIC6.

order

Optional sort order indicator. Options are ascending order (A) or descending order (D). The default is ascending order.

If you omit the KEY parameter, the merge uses one key; the key field extends from the beginning of the record to the smallest minimum record length (MINRL) defined for an input file; the key field has key type ASCII and is sorted in ascending order.

NOTE

If you intend to omit the KEY parameter, you should change the minimum record length attribute value for all input files. If you omit the KEY parameter and have not specified a minimum record length for all files, the merge attempts to use the default minimum record length as the key length. The default minimum record length is (0) zero; the merge cannot define a key of length 0 so it returns a fatal error.

DIRECTIVES or *DIRECTIVES_FILE* or *DIR* or *DF*

List of 1 through 100 directive files. If you omit the DIRECTIVES parameter, no parameters are read from a directive file; the merge is completely specified on the command.

LIST or *L*

Listing file. If you omit the LIST parameter, the merge writes listing information on file $LIST.

LIST_OPTIONS or *LO*

List of one or more list options specifying the additional information to be written to the listing file.

If you omit the LIST_OPTIONS parameter, the merge writes only the minimum information to the listing file (page number, error messages, directives, exception records file summary, and the number of records merged).

The valid keywords are:

OFF or NONE

No additional information is written to the listing file

S

Source directives read

DE

Detailed exception information (specify only when EXCEPTION_RECORDS_FILE parameter is specified)

RS

Record statistics

MS

Merge statistics

ERROR or *E*

Error message file. If you omit the ERROR parameter, the merge writes any error messages on file $ERRORS.

ERROR_LEVEL or *EL*

Minimum error severity to be reported. If you omit the ERROR_LEVEL parameter, the merge reports all warning, fatal, and catastrophic errors.

The valid keywords are:

INFORMATIONAL (I)

Report all errors

TRIVIAL (T)

Report all errors

WARNING (W)

Report warning, fatal and catastrophic errors only

FATAL (F)

Report fatal and catastrophic errors only

CATASTROPHIC (C)

Report catastrophic errors only

NONE

Report no errors

RESERVED_POSITION_9

Reserved.

ESTIMATED_NUMBER_RECORDS or *ENR*

Integer range from 1 through 16,777,215. You can specify the ESTIMATED_NUMBER_RECORDS parameter, but the merge does not use the parameter value.

EXCEPTION_RECORDS_FILE or *ERF*

File to which invalid records are written (invalid records are not written to the output file). If you omit this parameter, the merge does not perform exception processing; invalid records are written to the output file.

C170_COMPATIBLE or *CC*

Specifies whether lowercase letters in owncode procedure names are to be converted to uppercase letters. required for loading of the owncode procedure. If you omit this parameter, the default is OFF and the owncode procedure names are not converted. Use YES, TRUE, or ON to specify a logical true.

OMIT_DUPLICATES or *OD*

Specifies whether merge outputs only one record in each set of records with equivalent key values. Duplicates are not omitted; equivalent key values are processed as specified by the OWNCODE_PROCEDURE_5, RETAIN_ORIGINAL_ORDER, or SUM parameter.

TRUE, YES, or NO

Duplicates are omitted.

FALSE, NO, or OFF

Duplicates are not omitted.

OWNCODE_FIXED_LENGTH or *OWNFL* or *OFL*

Fixed record length (integer from 1 through 4,096). If you omit the OWNCODE_FIXED_LENGTH parameter, the merge uses the OWNCODE_MAXIMUM_RECORD_LENGTH parameter value as the record length. Specification of this parameter or the OWNCODE_MAXIMUM_RECORD_LENGTH parameter is required if the TO and FROM parameters are omitted.

OWNCODE_MAXIMUM_RECORD_LENGTH or *OWNMRL* or *OMRL*

Maximum record length (integer from 1 through 4,096). If you omit the OWNCODE_MAXIMUM_RECORD_LENGTH parameter, the merge uses the record length attribute of the first input file read or output file written as the maximum record length. Specification of this parameter or the OWNCODE_FIXED_RECORD_LENGTH parameter is required if the TO and FROM parameters are omitted.

OWNCODE_PROCEDURE_1 or *OP1* or *OWN1*

Reserved.

OWNCODE_PROCEDURE_2 or *OP2* or *OWN2*

Reserved.

OWNCODE_PROCEDURE_3 or *OP3* or *OWN3*

Object library entry point name for the procedure that postprocesses each output record. If you omit the OWNCODE_PROCEDURE_3 parameter, no user-defined output record postprocessing is performed.

OWNCODE_PROCEDURE_4 or *OP4* or *OWN4*

Object library entry point name for the procedure that can insert a record at the end of the output file. If you omit the OWNCODE_PROCEDURE_4 parameter, no user-defined output file postprocessing is performed.

OWNCODE_PROCEDURE_5 or *OP5* or *OWN5*

Object library entry point name for the procedure called if two input records have equal key values. This parameter cannot be specified with the SUM parameter. If you omit the OWNCODE_PROCEDURE_5 parameter, no user-defined processing of equal records is performed.

RETAIN_ORIGINAL_ORDER or *ROO* or *RETAIN* or *RET*

Reserved.

COLLATING_SEQUENCE_NAME or *CSN* or *SEQN*

Name of the collating sequence defined by the subsequent collating sequence parameters. If you omit the COLLATING_SEQUENCE_NAME parameter, no user collating sequence is defined.

COLLATING_SEQUENCE_STEP or *CSS* or *SEQS*

List of one or more value step definitions. If you omit the COLLATING_SEQUENCE_STEP parameter, no specific collating sequence value steps are defined.

A value step definition is a value set that defines one or more value steps in the collating sequence as follows (char is an ASCII character):

('char')

One 1-character value step

('char','char',...)

One multicharacter value step

('char'..'char')

Multiple 1-character value steps

('char'..'char','char'..'char',...)

Multiple multicharacter value steps. (All ranges must be the same size.)

COLLATING_SEQUENCE_REMAINDER or *CSR* or *SEQR*

Indicates whether a special value step is defined containing all characters not specified by other value step definitions. If you omit the COLLATING_SEQUENCE_REMAINDER parameter, the remaining characters keep their same collating positions as in the default ASCII collating sequence.

COLLATING_SEQUENCE_ALTER or *CSA* or *SEQA*

Indicates whether all characters in a value step are altered to match the first character in the value step. If you omit the COLLATING_SEQUENCE_ALTER parameter, no character alteration is performed.

SUM or *S*

List of 1 through 100 sum field definitions. This parameter cannot be specified with the OWN5 or RETAIN parameters. If you omit the SUM parameter, no summing is performed.

A sum field definition is a value set containing up to four values as follows:

(first..last,type,repeat_count) or

(first,length,type,repeat_count)

first

Position of the first byte of the sum field within the record. (The leftmost byte in a record is numbered 1.)

last

Position of the last byte of the sum field within the record.

length

Optional number of bytes in the sum field. The default length is 1 byte.

type

Name of a numeric data format. The default format is INTEGER. The valid numeric data format names are:

BINARY
BINARY_BITS
INTEGER
INTEGER_BITS
NUMERIC_FS
NUMERIC_LO
NUMERIC_LS
NUMERIC_NS
NUMERIC_TO
NUMERIC_TS
PACKED
PACKED_NS

repeat_count

Optional integer specifying the number of consecutive sum fields defined by the value set. The default number if 1.

ZERO_LENGTH_RECORDS or *ZERO_LENGTH_RECORD* or *ZLR*

Specifies the disposition of zero-length input records. This parameter applies only to records read from input files; it does not apply to records supplied by owncode procedures.

The keywords specifying the disposition of all zero-length input records read for the sort are as follows:

DELETE

Each zero-length record is deleted from the sort or merge. It is not written to the exception records file.

PAD

Each zero-length record is processed as a short record.

LAST

Each zero-length record is written at the end of the output.

VERIFY_MERGE_INPUT_ORDER or *VERIFY* or *VMIO* or *VER*

Indicates that the merge should check that all merge input records are in sorted order. If you omit the VERIFY_MERGE_INPUT_ORDER parameter, the merge does not check the input record order.

LOAD_COLLATING_TABLE or *LCT*

Loads a collating table to be used by one or more keys. The parameter specifies two values enclosed in parentheses. The first value is the key type name you specify in the key field definition. The second value is the name of the collation table. It can be one of the NOS/VE predefined collation table or a user-defined collation table.

RESULT_ARRAY or *RA* or *RESA*

Specifies an SCL variable in which merge statistics are returned. The variable must be a 16-element integer array. The first element of the array must be assigned the number of statistics to be returned (0 through 15).

Remarks

- A user-defined collating sequence can be defined by a sequence of parameters.
- The first parameter in the sequence must be a COLLATING_SEQUENCE_NAME parameter naming the collating seqence.
- The naming parameter is followed by one or more COLLATING_SEQUENCE_STEP, COLLATING_SEQUENCE_REMAINDER, and COLLATING_SEQUENCE_ALTER parameters defining the collating sequence steps that differ from the default ASCII collating sequence.
- The collating sequence definition ends when a parameter other than COLLATING_SEQUENCE_STEP, COLLATING_SEQUENCE_REMAINDER, or COLLATING_SEQUENCE_ALTER is read.
- More than one collating sequence can be defined for the sort.
- For more information, see the NOS/VE Advanced File Management Usage manual.

Examples

The following command merges the sorted records on files $USER.FILE3 and $USER.FILE5 and writes records on file $USER.FILE6. The records are merged in sorted order using the leftmost 10 characters as the key; the keys are sorted in ascending order using the ASCII collating sequence. The VERIFY parameter ensures that the input records are presorted.

```
/merge,($user.file3,$user.file5),$user.file6,..
../key=((1,10)),verify=yes.
```

$MESSAGE_LEVEL Function

Purpose

Returns a boolean value that identifies the message level currently selected for messages produced in your job.

For more information about setting the message level for a job, refer to the SET_MESSAGE_MODE command.

Format **$MESSAGE_LEVEL**
(keyword)

Parameters **keyword**

Specifies the message level you want verified. Use one of the following keywords:

BRIEF (B)

FULL (F)

If the message level specified is the same as the current message level, TRUE is returned. If the message level specified is different from the current level, FALSE is returned.

Remarks For further information about functions, see the NOS/VE System Usage manual.

Examples The following example indicates that BRIEF is the message level selected for messages within your job:

```
/display_value $message_level(brief)
TRUE
```

$MIN_INTEGER
Function

Purpose Returns the minimum integer value allowed for a parameter.

Format **$MIN_INTEGER**

Parameters None.

Remarks

- The value returned is 9,223,372,036,854,775,807
- For further information about functions, see the NOS/VE System Usage manual.

Examples In the following example, the system is interrogated for the value of $MIN_INTEGER:

```
/display_value $min_integer
-9223372036854775807
```

$MOD Function

Purpose Returns the modulus of one integer with respect to another integer.

Format

$MOD
(integer1
integer2)

Parameters

integer1

Specifies the number for which the modulus is to be returned. This parameter is required.

integer2

Specifies the number to which the modulus is relative. This parameter is required.

Remarks

- The result is calculated using the following formula (*a* and *b* are integers, and the division is integer division):

  modulus(a,b) = a-((a/b)*b)

- For further information about functions, see the NOS/VE System Usage manual.

Examples The following example returns the modulus of 134 with respect to 10.

```
/display_value $mod(134,10)
4
```

$NAME Function

Purpose Converts a string to an SCL name.

Format

$NAME
(string)

Parameters

string

Specifies the string you want converted to an SCL name. This parameter is required.

Remarks

- With this function, you can construct a name using a string and later use it as a name parameter value.
- The result of this function is convertd to uppercase characters.
- For further information about functions, see the NOS/VE System Usage manual.

Examples

- The following example creates a string variable named DATE_FORMAT. This value is converted to a name and passed to the $DATE function as the date option.

```
/date_format = 'month'
/display_value date_format
month
/display_value $name(date_format)
MONTH
/display_value $date($name(date_format))
March 28, 1987
```

- The following example illustrates the result when an invalid name is supplied to the $NAME function:

```
/date_format = 'mon th'
/display_value $date($name(date_format))
--ERROR-- Improper name: mon th
```

$NATURAL_LANGUAGE Function

Purpose Returns a name indicating the natural language of messages produced within your job.

Format $NATURAL_LANGUAGE

Parameters None.

Remarks

- Possible results are as follows:

DANISH	ITALIAN
DUTCH	NORWEGIAN
ENGLISH	PORTUGUESE
FINNISH	SPANISH
FLEMISH	SWEDISH
FRENCH	US_ENGLISH
GERMAN	Any other name

- For more information about specifying the language for messages, refer to the CHANGE_NATURAL_LANGUAGE command.
- For further information about functions, see the NOS/VE System Usage manual.

Examples The following example indicates that American English is the language of the job's messages:

```
/display_value $natural_language
US_ENGLISH
```

NETWORK_OPERATOR_UTILITY Command

Remarks Reserved for site personnel, Control Data, or future use.

OPEN_FILE_MIGRATION_AID Command

Purpose Calls the File Migration Aid (FMA) and opens the file migration environment.

Format **OPEN_FILE_MIGRATION_AID** or **OPEFMA**

PARTNER_JOB_CARD=string
STATUS=status variable

Parameters *PARTNER_JOB_CARD* or *PJC*

Specifies the job statment parameters for the CYBER 170 partner job. The specified string must conform to either NOS (for NOS dual state sytems) or NOS/BE (for NOS/BE dual state systems) job statement syntax. Omission causes a default job statement to be used. If the NOS/VE job is a batch job, the default job statement has an infinite time limit with no other parameters specified; if the NOS/VE job is an interactive job, the default job statement has no parameters specified.

Remarks

- The OPEN_FILE_MIGRATION command initiates a batch job, called the partner job, on the CYBER 170 (NOS or NOS/BE) side of the dual state system.
- An OPEN_FILE_MIGRATION command must be executed before any other FMA command can be entered.

Examples The following example opens the File Migration Aid, enters several FMA commands to migrate a NOS FORTRAN file, and closes the File Migration Aid.

```
/open_file_migration_aid
fa/open_170_state
fa/execute_command 'attach,binfile'
fa/execute_command 'file,binfile,fa=sq,rt=w,bt=i,mrl=90.'
fa/close_environment
fa/execute_migration_task migration_file=(binfile, ..
fa../newfile,c170_to_c180) file=lgo
fa/close_environment
```

OPERATE_STATION Command

Remarks Reserved for site personnel, Control Data, or future use.

$ORD Function

Purpose Returns the integer that corresponds to the ASCII character you specify.

Format

$ORD
(string)

Parameters

string

Specifies the ASCII character for which you want the ordinal returned. This parameter is required.

Remarks

- The integer returned is an ordinal; it represents the position of the character in question within the ASCII collating sequence.
- For further information about functions, see the NOS/VE System Usage manual.

Examples

- The following example returns the decimal integer that corresponds to the ASCII character '#':

```
/display_value $ord('#')
35
```

- The following example performs the same operation, but displays the hexadecimal value of the character ordinal:

```
/post_radix = '(16)'
/display_value $strrep($ord('#'),16)//post_radix
23(16)
```

$OUTPUT_STATUS Function

Purpose Returns information about the status of an output file.

Format

```
$OUTPUT_STATUS
    (name,
    keyword)
```

Parameters

name

Specifies the file for which you want status returned. Use either a system-supplied or user-supplied file name. If the name is user-supplied, it must be unique; otherwise the function will fail. This parameter is required.

keyword

Specifies the type of information you want returned.

Chapter 3, Function Attributes, lists and describes the keyword values you can supply and the corresponding function results. This parameter is required.

Remarks

- You can use the OUTPUT_STATE attribute to test for the existence of a particular output file.
- The type of result returned depends on the attribute being tested.
- For further information about functions, see the NOS/VE System Usage manual.

Examples The following example displays the status of file xyz:

```
/print_file f=xyz
/display_value $output_status(xyz,state)
QUEUED
```

$PARAMETER Function

Purpose Returns the evaluated form of the specified parameter value list.

Format

$PARAMETER
(name)

Parameters

name

Specifies the parameter for which you want the value list returned. This parameter is required.

Remarks

- This function is used to reference parameters within procedures.
- For further information about functions, see the NOS/VE System Usage manual.

Examples The example is based on the following procedure header:

```
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
    )
```

Consider the following call to the preceding procedure:

```
/display_number (4,(2..5,2),1*100(2),5*6) out_file
```

In this case, each element in the value list for the first parameter is evaluated and displayed by including the following command in the DISPLAY_NUMBER procedure:

```
display_value $parameter(number)
```

It writes the following value list to the output file:

```
(4, (2..5,2), 4, 30)
```

$PARAMETER_LIST Function

Purpose Returns the entire unevaluated parameter list for the procedure call.

Format **$PARAMETER_LIST**

Parameters None.

Remarks

- This function is used to reference parameters within procedures.
- The parameter list is returned as a string.
- For further information about functions, see the NOS/VE System Usage manual.

Examples The example is based on the following procedure header:

```
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
    )
```

Consider the following call to the preceding procedure:

```
/display_number (1,(2..3,2),5,6) out_file
```

The entire parameter list is displayed by including the following command in the DISPLAY_NUMBER procedure:

```
display_value $parameter_list
```

It writes the following information to the output file:

```
(4,(2..3,2),5,6) out_file
```

PASCAL Command

Purpose Calls the PASCAL compiler, specifies the files to be used for input and output, and indicates the type of output to be used.

Format

PASCAL
INPUT=file
BINARY=file
LIST=file
ERROR=file
ERROR_LEVEL=keyword
LIST_OPTIONS=list of keyword
OPTIMIZATION_LEVEL=keyword
DEBUG_AIDS=list of keyword
RUNTIME_CHECKS=list of keyword
STANDARDS_DIAGNOSTICS=list of keyword
TERMINATION_ERROR_LEVEL=keyword
STATUS=status variable

Parameters

INPUT or *I*

File that contains the source text to be read. If omitted, $INPUT is assumed. I=$NULL results in termination of the compilation.

BINARY or *B*

File on which object code is written. If omitted, $LOCAL.LGO is assumed. If $NULL is specified, the compiler performs a syntactic and semantic scan of the program, but does not generate any object code.

LIST or *L*

File on which the source (compilation) listing, diagnostic listing, object listing, statistics and reference attributes are written.

If omitted, $LIST is assumed. If $NULL is specified, all compile-time output is discarded.

ERROR or *E*

File to which Pascal writes the text of diagnostic messages. Diagnostic messages are also written to the LIST file if present. If ERROR and LIST specify the same file, only one copy of the diagnostics is output. If omitted, $ERROR is assumed. If $NULL is specified, no error file is written.

ERROR_LEVEL or *EL*

Any of the following error list options.

C

Lists only catastrophic diagnostic messages.

F

Lists only fatal diagnostic messages.

W

Lists warning (informative) diagnostic messages as well as fatal diagnostic messages.

LIST_OPTIONS or *LO*

Combination of the following options about information that appears in the LIST file. If omitted, option S is assumed.

A

Produces an attribute list of each entity in the program. The attribute listing is produced following the source listing on the file specified by the LIST parameter or, if the LIST parameter is omitted, on file $LIST. If the R option is selected, the references are shown on the same listing.

O

Lists compiler-generated object code. When selected, this listing includes an assembly-like listing of the generated object code. This option has no effect if the BINARY_OBJECT parameter is set to $NULL.

R

Produces a symbolic cross-reference listing that shows the location of all program entities and their use within the program.

S

Lists the source input file.

NONE

No listing is printed.

OPTIMIZATION_LEVEL or *OL*

Any of the following optimization options. If omitted, LOW is assumed.

DEBUG

Object code is stylized to facilitate debugging. Stylized code contains a separate packet of instructions for each executable source statement; it carries no variable values across statement boundaries in registers, and it notifies Debug each time the beginning of a statement or procedure is reached.

LOW

Provides for keeping constant values in registers.

DEBUG_AIDS or *DA*

Any of the following debug options. If omitted, NONE is assumed.

DT

Debug generates line number tables or symbol tables with the object code.

ALL

Debug selects all of the available options.

NONE

Debug line number table, and Debug symbol table are not generated with the object code.

RUNTIME_CHECKS or *RC*

A combination of the following run-time checking options are compiled into the object program. If omitted, NONE (no run-time checks are produced) is selected.

F

Selects checking of errors involving file variables and buffer variables.

N

Selects checking of misuse of pointer variables and buffer variables, and invalid usage of NEW and DISPOSE procedures.

R

Selects range checking for subrange and set assignments and case variables.

S

Selects array subscript bound checking.

T

Selects checking of variant tag fields.

ALL

Selects all of the run-time checks available.

NONE

Selects no run-time checks.

STANDARDS_DIAGNOSTICS or *SD*

Specifies whether the use of nonstandard extensions in a program is to be diagnosed. The first option (LEVEL) defines the error level to be assumed by such diagnostics. The second option (STANDARD) determines which of two standards is used. If omitted, NONE is selected and nonstandard extensions are not diagnosed.

(W)

ISO standard errors result in warning errors.

(F)

ISO standard errors result in fatal errors.

(W,ISO)

ISO standard errors result in warning errors.

(F,ISO)

ISO standard errors result in fatal errors.

(W,ANSI)

ANSI standard errors result in warning errors.

(F,ANSI)

ANSI standard errors result in fatal errors.

(NONE)

Standard errors are not diagnosed.

TERMINATION_ERROR_LEVEL or *TEL*

Indicates the diagnostic severity level at which the PASCAL compiler returns an abnormal STATUS. If omitted, F (fatal level diagnostics return an abnormal STATUS) is selected.

W

Warning level and higher diagnostics return an abnormal STATUS.

F

Fatal level diagnostics return an abnormal STATUS.

C

Catastrophic level diagnostics return an abnormal STATUS.

Remarks

- If the INPUT parameter specifies file $INPUT, you will be prompted with a question mark (?) for one line of source code at a time. Enter the END_OF_INFORMATION value (*EOI) to terminate and compile your input.

  The EOI value is a connection attribute defined by your site administrator (NOS/VE default is *EOI). To display the value for your site, enter:

  ```
  /display_term_conn_default, end_of_information
  ```

  For more information on connection attributes, see the NOS/VE System Usage manual.

- For more information, see the PASCAL for NOS/VE manual.

Examples

This command reads source code from a file named SOURCE_REPORT, writes the listing file on the file LIST_FILE, and writes the object code on the file OBJECT_REPORT. The listing includes source code, compiler-generated object code, and a symbolic cross-reference listing. ISO standard errors result in only warning errors. Fatal level errors only result in an abnormal STATUS. All run-time error checks are selected.

The following default parameters have been selected:

DEBUG_AIDS
(D=NONE)

ERROR
(E=$ERRORS)

ERROR_LEVEL
(EL=W)

The command:

```
/pascal i=source_report l=list_file ..
../b=object_report lo=(o,r) sd=w tel=f rc=all
```

$PATH Function

Purpose Returns either a portion of a path (a string) or the count of the elements in a path (an integer).

Format

$PATH
(file
keyword)

Parameters

file

Specifies the file path you are querying. This parameter is required.

keyword

Specifies whether a portion of the path or a count of the elements is to be returned. This parameter is required. Use one of the following names:

CATALOG

Causes the catalog portion of the path to be returned as a string.

LAST

Causes the last element of the path to be returned as a string.

COUNT

Returns an integer count of the elements contained in the path.

Remarks

- When a string value is returned by the function, all letters within the string are uppercase.
- For further information about functions, see the NOS/VE System Usage manual.

Examples

- The following example displays the catalog portion of the file path USER.DATA_FILE_1. The user name is USER_123 and the family name is FAMILY_Z.

```
/display_value $path($user.data_file_1,catalog)
:FAMILY_Z.USER_123
```

- The following example displays the last element in the preceding file path:

```
/display_value $path($user.data_file_1,last)
DATA_FILE_1
```

- The following example displays the number of elements in the preceding file path. The count of three is derived from the family name FAMILY_Z, user name USER_123, and file name DATA_FILE_1.

```
/display_value $path($user.data_file_1,count)
3
```

- See the online EXAMPLES manual for a procedure demonstrating how you can extract the different parts of a file path. The procedure's name is SHOW_FILE_PATH.

PHYSICAL_CONFIGURATION_UTILITY Command

Remarks Reserved for site personnel, Control Data, or future use.

POP Control Statement

Purpose Deletes a current version of a system environment object in an SCL procedure, and restores the changed environment object to its former value.

Format **POP**
environment object(s)

Parameters **environment objects**

Specifies one or more components of the system environment (environment objects) to be deleted. Multiple components must be separated by spaces or commas. An error occurs if the environment object was not previously established (pushed). Choose from the following list of system environment objects:

COMMAND_LIST
FILE_CONNECTIONS
INTERACTION_STYLE
MESSAGE_LEVEL
NATURAL_LANGUAGE
PROGRAM_ATTRIBUTES
WORKING_CATALOG

Remarks For more information, see the NOS/VE System Usage manual.

$PREVIOUS_STATUS Function

Purpose Returns the completion status of the previous command.

Format **$PREVIOUS_STATUS**

Parameters None.

Remarks For further information about functions, see the NOS/VE System Usage manual.

Examples The example shows is based on the following procedure:

```
/collect_text display_status
ct? proc display_status, diss(
ct?   status:status=$previous_status)
ct?   display_value $value(status)
ct? procend display_status
ct? **
/
```

If the system cannot find the status message associated with the status condition, the following message is displayed:

```
/display_value $status(false, 'MY',0,'param1','param2')
--ERROR--  CC=MY 0 TEXT=?param1?param2
```

If you attempt to create a duplicate variable, the $PREVIOUS_STATUS function can obtain the status of the previous command, as in the following example:

```
/create_variable s kind=status
/create_variable x status=s
/display_status
NORMAL STATUS
/create_variable x status=s
/display_status
--ERROR-- X is already declared as a variable.
```

PRINT_FILE Command

Purpose Schedules one or more files for printing.

Format

PRINT_FILE or
PRINT_FILES or
PRIF
FILE=list of file
COMMENT_BANNER=string
COPIES=integer
DATA_MODE=keyword
DEVICE=name or *keyword*
EXTERNAL_CHARACTERISTICS=string or *keyword*
FORMS_CODE=string or *keyword*
OPERATOR_FAMILY=name
OPERATOR_USER=name
OUTPUT_CLASS=keyword
OUTPUT_DESTINATION=any
OUTPUT_DESTINATION_USAGE=name or *keyword*
OUTPUT_PRIORITY=keyword
REMOTE_HOST_DIRECTIVE=string
ROUTING_BANNER=string
STATION=name or *keyword*
USER_FILE_NAME=list of name
VERTICAL_PRINT_DENSITY=keyword
VFU_LOAD_PROCEDURE=name or *keyword*
STATUS=status variable

Parameters

FILE or **FILES** or **F**

Specifies the files to be printed. This parameter is required.

COMMENT_BANNER or *CB*

Specifies a character string to be displayed with the printed file. Use of this string is determined by the site.

If omitted, the COMMENT_BANNER job attribute is used. If the COMMENT_BANNER attribute is an empty string, the file name is used.

COPIES or *C*

Specifies the number of copies of the printout required. Omission causes the COPIES job attributes to be used.

DATA_MODE or *DM*

Specifies the data mode of the file to be printed. You can specify the following keyword values:

CODED

Specifies that the data in the file contains codes to be interpreted and handled by the printer.

TRANSPARENT

Specifies that the data in the file should be printed without conversion or interpretation. This keyword cannot be used if the OUTPUT_DESTINATION_USAGE attribute is DUAL_STATE.

DEVICE or *D*

Specifies a name that, when combined with the STATION attribute value, identifies the printer at which the file is to be printed. Values 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 specified. If omitted, the device job attribute is used.

EXTERNAL_CHARACTERISTICS or *EC*

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 for this parameter 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.

If the file is being sent through QTF to a NOS system for printing, NORMAL is mapped to the NOS A9 value. For more information on the NOS A9 value, see the NOS 2 Reference Set, Volume 3.

If omitted, the EXTERNAL_CHARACTERISTICS job attribute is used.

FORMS_CODE or *FC*

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 parameter is defined by the site.

Values for this parameter can be any string of 1 to 6 characters or the keyword NORMAL. If you specify NORMAL, the system selects a printer that has a FORMS_CODE value of NORMAL. If you specify NORMAL when the OUTPUT_DESTINATION_USAGE attribute is DUAL_STATE, the NORMAL value is equivalent to a string of spaces. When OUTPUT_DESTINATION_USAGE is PUBLIC or PRIVATE, keyword NORMAL is equivalent to the string 'normal'.

If omitted, the FORMS_CODE job attribute is used.

OPERATOR_FAMILY or *DESTINATION_FAMILY* or *DF* or *OF*

Specifies the family name of a private station or remote system operator. This family name together with the OPERATOR_USER parameter identifies the private station operator or remote system operator who can print or receive the file. This parameter is also used to establish the control family attribute of the output file. This parameter is not meaningful unless the OUTPUT_DESTINATION_USAGE attribute is PRIVATE or NTF.

OPERATOR_USER or *STATION_OPERATOR* or *SO* or *OU*

Specifies the user name of a private station or remote system operator. This user name together with the OPERATOR_FAMILY parameter identifies the private station operator or remote system operator who can print

or receive the file. This parameter is also used to establish the control user attribute of the output file. This parameter is not meaningful unless the OUTPUT_DESTINATION_USAGE attribute is PRIVATE or NTF.

OUTPUT_CLASS or *OC*

Specifies an output class for the output file. The output class defines the initial priority, the maximum priority, an aging interval, and an aging factor for the output file.

For this release, 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 one second, and an aging factor of one priority unit per aging interval.

If omitted, the OUTPUT_CLASS job attribute is used.

OUTPUT_DESTINATION or *ODE*

Specifies the location name of the system where the output file is to be sent for printing if the file's OUTPUT_DESTINATION_USAGE attribute is QTF or NTF. For all other values of OUTPUT_DESTINATION_USAGE, this parameter is not meaningful and 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. For more information, see your Site Administrator.

If this parameter is omitted, the OUTPUT_DESTINATION job attribute is used.

OUTPUT_DESTINATION_USAGE or *DESTINATION_USAGE* or *DU* or *ODU*

Specifies either the kind of CDCNET print station where the file is to be printed, or the queue file transfer application to be used to forward the output file to a remote system. The following options are available:

PUBLIC

Indicates that the file is to be printed at a public CDCNET batch I/O station. If this value is specified, the OPERATOR_FAMILY, OPERATOR_USER, OUTPUT_DESTINATION, and REMOTE_HOST_DIRECTIVE attributes are not meaningful.

PRIVATE

Indicates that the file is to be printed at a private CDCNET batch I/O station when the designated station operator is controlling the station. If this value is specified, the OUTPUT_DESTINATION and REMOTE_HOST_DIRECTIVE attributes are not meaningful.

DUAL_STATE

Indicates that the file is to be printed under control of the dual-state partner system. The NOS/VE file is copied to a NOS or NOS/BE queue file in the 12-bit ASCII format with zero-byte terminated (Z-type) records. If this value is specified, no other attributes are meaningful with the exception of the FORMS_CODE, COPIES, ROUTING_BANNER, and REMOTE_HOST_DIRECTIVE attributes.

QTF

Indicates that the file is to be forwarded to the remote system identified by the OUTPUT_DESTINATION attribute for processing by that system.

NTF

Indicates that the file is to be forwarded to the remote NTF system identified by the OUTPUT_DESTINATION attribute for processing by that system.

If this parameter is omitted, the OUTPUT_DESTINATION_USAGE job attribute is used unless the value of the OUTPUT_DISPOSITION job attribute is LOCAL. If the OUTPUT_DISPOSITION job attribute is LOCAL, the system default is used.

OUTPUT_PRIORITY or *OP*

Specifies a priority increment that is added to the output file's initial priority (defined by the output class).

Keywords are:

Keyword	Increment
LOW	0
MEDIUM	1500
HIGH	3000

If omitted, the job's default attribute is used.

REMOTE_HOST_DIRECTIVE or *DUAL_STATE_ROUTE_PARAMETERS* or *DSRP* or *RHD*

Specifies a default text string which may be used to control output processing of output files. 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.

This parameter is ignored unless the OUTPUT_DESTINATION_USAGE output attribute specify the appropriate value. For more information on submitting output files to remote systems, see the NOS/VE System Usage manual.

If omitted, the REMOTE_HOST_DIRECTIVE job attribute is used.

ROUTING_BANNER or *RB*

Specifies a character string to be displayed with the printed file. The actual use of this string is determined by the site. If omitted, the ROUTING_BANNER job attribute is used. If that attribute is an empty string, the control user name for the file is used.

STATION or *S*

Specifies the 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 you specify AUTOMATIC, the system default is used.

If omitted, the STATION job attribute is used.

USER_FILE_NAME or *USER_FILE_NAMES* or *UFN*

Specifies a list of names to be associated with the files being printed. The names in this list are matched positionally with the files specified in the FILE parameter (the first name in the list is matched with the first file to be printed, and so on). For any file being printed, if a user-supplied name is not specified, the file name is used.

VERTICAL_PRINT_DENSITY or *VPD*

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.

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.

If this parameter is omitted, the VERTICAL_PRINT_DENSITY job attribute is used.

VFU_LOAD_PROCEDURE or *VLP*

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 parameter 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.

If this parameter is omitted, the VFU_LOAD_PROCEDURE job attribute is used.

Remarks

- The USER_FILE_NAME and DATA_MODE parameters of the PRINT_FILE command are the only parameters whose defaults are not based on your job's attributes.
- If the OUTPUT_DISPOSITION job attribute is DISCARD_ALL_OUTPUT or WAIT_QUEUE, the file submitted for output by the PRINT_FILE command will not be printed. See the CHANGE_JOB_ATTRIBUTE command for more information on the OUTPUT_DISPOSITION job attribute.
- A file is processed according to the value of its output attributes at the time it leaves the output queue. Refer to the CHANGE_OUTPUT_ATTRIBUTE command for information on how to change a file's output attributes while it is in the output queue.
- Transfers to non-NOS/VE systems are not currently supported.
- For more information, see the NOS/VE System Usage manual.

Examples

The following example prints five copies of a listing.

```
/print_file file=list copies=5
/display_output_status all
Output_State                : printing
System_File_Name            : $0990_0102_aad_1439
User_File_Name              : list
/disos name=all
 None Were Found.
```

The following example illustrates the use of PRINT_FILE using the REMOTE_HOST_DIRECTIVE parameter to print the file FORTRAN_LISTING on the partner system.

```
/print_file file=fortran_listing ..
../output_destination_usage=dual_state ..
../remote_host_directive='dc=pr,ec=a9'
```

To print the file ANY_OUT at the remote mainframe VN3, enter:

```
/print_file f=any_out ode=vn3 odu=qtf
```

The file in the preceding example will be printed using the remote mainframe's default job attributes. To specify job attributes other than the remote mainframe's default job attributes, you must use the REMOTE_HOST_DIRECTIVE parameter. See the REMOTE_HOST_DIRECTIVE parameter description earlier in this manual for more information.

PRINT_LETTER Command

Remarks Reserved for site personnel, Control Data, or future use.

PROC Control Statement

Purpose Defines the names by which an SCL procedure can be called, the procedure attributes, and the parameters to the procedure.

Format **PROC list of procedure name** *(*
list of parameter definition)

Parameters **procedure name**

Specifies the name by which procedure is to be called. Any one of the list of procedure names may be used as the command name to call the procedure. This parameter is required.

parameter definition

Defines the procedure parameters. A parameter is defined as follows:

parameter names : value specification = default specification

Remarks

- If the value specification is omitted for a parameter, FILE is assumed unless the parameter name is STATUS, in which case a status variable is assumed.
- If you specify $REQUIRED as the default specification for a parameter, the user must enter a value for that parameter. If you specify $OPTIONAL as the default specification for a parameter, or if you omit the default specification, the user can optionally enter a parameter value.
- The value specification specifies the type of value and whether or not it can be represented as a list and/or range. The value specification is comprised of the following elements:

  data type

  Specifies the type of value the parameter can be.

  value list type

  Specifies whether the parameter value can be given as a list and range of values.

- The data type clause defines the type of value, whether it is a variable or array, and whether it can be represented by one or more keywords. The following are the formats of the data type specification.

  data type
  data type OR KEY keywords
  VAR OF variable type
  VAR OF variable type OR KEY keywords
  ARRAY OF variable type
  ARRAY OF variable typr OR KEY keywords

  The following table lists the data types:

Data Type	Description
FILE	Specifies a file.
NAME	Specifies an SCL name.
STRING	Specifies a string.
INTEGER	Specifies an integer.
REAL	Specifies a real variable.

KEY	Keyword value.
BOOLEAN	Specifies a boolean value.
STATUS	Specifies a status variable.
ANY	Specifies that any data type can be used.
application value name	Specifies the name of an application value. For more information on application procedures, see the CYBIL System Interface manual.

- The following formats of the value list type are valid:

  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

  The value list type defines the parameter as a list of value sets. The number of value sets allowed is specified with the value set count clause.

- For more information, see the NOS/VE System Usage manual.

Examples

The following example defines a procedure named DISPLAY_NUMBER. The procedure has the alternate names DISPLAY_NUMBERS and DISN. It accepts a parameter named NUMBER, which can be represented by from 1 through 10 value sets that must be of kind integer. Each value set can contain from 1 through 2 value elements. Each value element can be specified as a range. The NUMBER parameter is required and has the alternate names NUMBERS and N.

```
proc display_number,display_numbers,disn (
    number,numbers,n : list 1..10, 1..2, ..
        range of integer = $required
    status)
```

PROCEND
Control Statement

Purpose Terminates an SCL procedure. Execution of a PROCEND causes normal status to be returned by the procedure.

Format **PROCEND** *procedure name*

Parameters **procedure name**

If the procedure name is given, it must be identical to the first procedure name defined in the PROC statement for the procedure being terminated. This is for checking purposes only and does not affect the meaning of the statement.

Remarks For more information, see the NOS/VE System Usage manual.

Examples The following procedure is defined.

```
PROC display_number,display_numbers,disn (
    number,numbers,n : list 1..10, 1..2, ..
        range of integer = $required
    status)
        .
        .
        .
PROCEND display_number
```

$PROCESSOR
Function

Purpose Returns the specified attribute of a hardware processor in the mainframe on which the request is made.

Format **$PROCESSOR**
(keyword
integer)

Parameters **keyword**

Specifies the attribute you want returned. This parameter is required. Use one of the following entries:

CLOCK (C)

Returns the integer value of the processor's free-running microsecond clock. When this keyword is specified, the integer parameter is ignored.

MODEL_TYPE (MT or MODEL or M)

Returns a string indicating the performance class of the mainframe's processors. The following are the possible values returned for this keyword:

CYBER 810 Class
CYBER 815 Class
CYBER 825 Class
CYBER 830 Class
CYBER 835 Class
CYBER 840 Class
CYBER 840S Class
CYBER 845 Class
CYBER 845S Class
CYBER 850 Class
CYBER 855 Class
CYBER 855S Class
CYBER 860 Class
CYBER 870 Class
CYBER 930 Class
CYBER 990 Class
CYBER 995 Class

MODEL_NUMBER (MN)

Returns a string that designates the processor's model number. The following are possible values returned for this keyword:

810
815
825
830
835
840
840S
845
845S

850
855
855S
860
870
9301
9303
990
990E

SERIAL_NUMBER (SERIAL or SN)

Returns a string value indicating the processor's serial number, for example, 109.

STATE (S)

Returns a string value indicating the processor's state; the states are ON, OFF, and DOWN.

integer

Specifies the processor whose attribute you want returned; applies only to a multiprocessor mainframe. The processor number starts at 0.

If the processor you specify does not exist, a null string is returned for all attributes except CLOCK.

If you do not specify a processor number, the number of the current processor is used.

Remarks For further information about functions, see the NOS/VE System Usage manual.

Examples The following procedure queries the system for the number of processors, model types, serial numbers, and model numbers.

```
PROC processor
  FOR i=1 TO $mainframe(total_processors) DO
    display_value 'Processor  '//$strrep(i-1)
    display_value ..
' Model Type      - '//$processor(model_type,i-1)
    display_value ..
' Serial Number   - '//$processor(serial_number,i-1)
    display_value ..
' Model Number    - '//$processor(model_number,i-1)
  FOREND
PROCEND processor
```

When the procedure is called, the following result is returned:

```
Processor 0
 Model Type      - CYBER 995 Class
 Serial Number   - 102
 Model Number    - 990
Processor 1
 Model Type      - CYBER 995 Class
 Serial Number   - 103
 Model Number    - 990
/
```

$PROGRAM Function

Purpose Returns the default program attributes for the job.

Format

$PROGRAM
(keyword1
keyword2)

Parameters **keyword1**

Specifies the program attribute you want returned. Chapter 3, Function Attributes, lists and describes the keyword values you can supply and the corresponding function results. This parameter is required.

keyword2

Specifies one of the keywords that you must include if you specify the LOAD_MAP_OPTION parameter. See the Remarks section for details.

Remarks

- The kind of result returned depends on the program attribute being tested. When a string value is returned, all letters are converted to uppercase.
- When LOAD_MAP_OPTION is supplied as the program attribute name, the format for $PROGRAM changes as follows:

  $PROGRAM
  (LOAD_MAP_OPTION)
  (keyword)

keyword

One of the keyword values listed in the following table. This parameter is required.

Keyword Value	Description
BLOCK (B)	Block map.
CROSS_REFERENCE (CR)	Entry point cross-reference map.
ENTRY_POINT (EP)	Entry point map.
NONE (N)	No load map.
SEGMENT (S)	Segment map.

- For more information about program attributes, see the NOS/VE Object Code Management manual.
- For further information about functions, see the NOS/VE System Usage manual.

Examples The following example tests whether debug mode is currently in effect:

```
IF $program(debug_mode) THEN
    .
    ."Perform special processing if in debug mode."
    .
IFEND
```

PROLOG Command

Purpose Calls the PROLOG interpreter.

Format

PROLOG
WORKSPACE=file
PASSWORD=name
WAIT=boolean
INPUT=file
OUTPUT=file
LIST_OPTION=list of keyword
QUESTION=string
STATUS=status variable

Parameters

WORKSPACE or *WS*

Specifies the name of the file containing a Prolog program (saved state). The saved state is restored when Prolog is activated. The file must have the following attributes:

- file_processor='prolog'
- file_organization=byte_addressable
- file_contents=object
- file_structure=data

The file is accessed with the following permits:

- access_mode=read
- share_mode=read

If omitted, WS=$SYSTEM.PROLOG.INITIAL_STATE, which contains all the system-supplied predicates, is selected.

PASSWORD or *PW*

Specifies the password required to access a file specified by the WORKSPACE parameter.

WAIT or *W*

Forces the Prolog interpreter to wait for the file specified by the WS parameter if that file is busy.

If WAIT=FALSE and the file is busy, the interpreter returns a non-normal status.

If omitted, WAIT=TRUE.

INPUT or *I*

Specifies the file that contains the Prolog statements to be read. This file is obtained from the terminal when 'user' is referenced in a Prolog session. For interactive use, I=$INPUT specifies the user's terminal.

I=$INPUT

Specifies interactive processing from your terminal.

I=file

Specifies batch processing.

If omitted, I=$INPUT is selected.

OUTPUT or *O*

Specifies the file on which the output is written.

O=filename

Indicates that the file remains when the job is completed.

O=$OUTPUT

Specifies your terminal if you are in interactive mode. If you are in batch mode, it specifies the file that is printed at job completion.

If omitted, O=$OUTPUT is selected.

LIST_OPTION or *LIST_OPTIONS* or *LO*

Specifies the information that is to be written to the output file. Options are:

B

Prevents the Prolog system banner from going to the output file.

P

Prevents prompts, including ?- and :-, from going to the output file.

S

Copies all input to the output file.

If LIST_OPTIONS is omitted and you are in interactive mode, no options are selected; otherwise, LO=(S,P) is selected.

Multiple options specified in the format:

LO=(op,..,op)

QUESTION or *Q*

Allows Prolog for NOS/VE to be started with a question. If omitted, Q='true' is selected.

Remarks For more information, see the PROLOG for NOS/VE manual.

Examples The first example shows an interactive session. The second example shows a batch session.

Interactive:

```
/prolog WS=prolog_example PW=shazam I=input O=output
```

The following default parameters are selected:

```
WAIT=TRUE
LIST_OPTIONS=no options
QUESTION='TRUE'
STATUS=no status available
```

Batch:

```
prolog I=cheese_report O=cheese_report_output
```

The following default parameters are selected:

```
WS=$SYSTEM.PROLOG.INITIAL_STATE
PASSWORD=no password
LIST_OPTIONS=(S,P)
QUESTION='TRUE'
STATUS=no status variable
```

PROLOG_UTILITY Command

Remarks Reserved for site personnel, Control Data, or future use.

PUSH Control Statement

Purpose Temporarily changes certain system environments in SCL procedures.

Format

PUSH
environment object(s)

Parameters

environment objects

Specifies one or more components of the system environment (environment objects) to be changed (pushed). The components must be separated by spaces or commas. Choose from the following list of system environment objects:

COMMAND_LIST
FILE_CONNECTIONS
INTERACTION_STYLE

MESSAGE_LEVEL
NATURAL_LANGUAGE
PROGRAM_ATTRIBUTES
WORKING_CATALOG

Remarks

- Only the most recently pushed version of the system environment can be referenced or changed.
- A specific object can be pushed only once in a procedure.
- For more information, see the NOS/VE System Usage manual.

Examples

The following example illustrates a procedure that 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 .ajl.command_library
      .
      .
    "Execute commands found in .ajl.command_library."
      .
      .
PROCEND
```

PUT_LINE Command

Purpose

Writes lines to a file.

Format

PUT_LINE or
PUT_LINES or
PUTL
LINES=list of string
OUTPUT=file
STATUS=status variable

Parameters

LINES or LINE or L

Specifies the lines to be written to the output file. This parameter is required.

OUTPUT or O

Identifies the output file to which lines are written. The default output file is $OUTPUT.

Remarks

- This command never adds page titling or format effectors. The system assumes the first character of each line is a format effector.
- 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.
- For more information, see the NOS/VE System Usage manual.

Examples

The following example writes a 3-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, 1987
The current time:   2:45 PM
Welcome to NOS/VE.
```

$QUEUE Function

Purpose Returns the state of a local queue.

Format

```
$QUEUE
   (name
   keyword)
```

Parameters

name

Name of the local queue you are interrogating. This parameter is required.

keyword

Specifies the attribute of the local queue you are interrogating. This parameter is required. The following are possible keywords:

CONNECT_COUNT
MESSAGE_COUNT
WAIT_COUNT

Remarks

- If you specify CONNECT_COUNT, the number of tasks connected to the queue is returned as an integer.
- If you specify MESSAGE_COUNT, the number of messages in the queue is returned as an integer.
- If you specify WAIT_COUNT, the number of tasks that are waiting for a message from the queue is returned as an integer.
- For further information about functions, see the NOS/VE System Usage manual.

Examples The following example interrogates the system for the number of messages in the local queue named WAIT:

```
/display_value $queue(wait,message_count)
0
```

QUICK Command

Purpose Enters the QUICK utility.

Format

QUICK
PROFILE=file
UID=dm_name
UPW=dm_name
ERROR=file
EXECUTE_ONLY=boolean
FROM=file
TO=file
LISTABLE_PROFILE=file
STATUS=status variable

Parameters

PROFILE or *P*

Specifies the profile. If omitted, PROFILE is assumed.

UID

Specifies the IM/DM user identification code. It can contain 8 letters and/or digits and is not enclosed in quotes.

UPW

Specifies the IM/DM user password. It can contain 8 letters and/or digits and is not enclosed in quotes.

ERROR or *E*

Specifies the file to which QUICK writes any error messages. If omitted, $LOCAL.QUICK_ERRORS is assumed.

EXECUTE_ONLY or *EO*

Specifies whether you are executing a previously prepared profile. If omitted, FALSE is assumed (you are executing in interactive mode).

FROM or *F*

Specifies the input data file. If omitted, QUICK obtains the file name from the INPUT_FILE_NAME window. This parameter is used only in EXECUTE_ONLY mode.

TO or *T*

Specifies the output file to which data is written. If omitted, QUICK obtains the file reference from the OUTPUT_NAME window. This parameter is used only in EXECUTE_ONLY mode.

LISTABLE_PROFILE or *LP*

Specifies the file which Quick uses to load the profile. The listable_profile is created by selecting the option "Write profile onto a file in listable format" on the utility screen of a previous execution of Quick. If omitted, Quick does not load the profile from a listable_profile.

Remarks For more information, see the IM/Quick manuals.

$QUOTE Function

Purpose Copies one string to another string and adds string delimiters.

Format

$QUOTE
(string)

Parameters

string

Specifies the string you want copied. This parameter is required.

Remarks For further information about functions, see the NOS/VE System Usage manual.

Examples The following example copies string S to string Q:

```
/s = 'ABC''DEF'
/display_value s
ABC'DEF
/q = $quote(s)
/display_value q
'ABC''DEF'
```

$RANGE Function

Purpose Returns a boolean value indicating whether a given parameter is specified as a range (low value to high value).

Format

$RANGE
(name
integer1
integer2)

Parameters

name

Specifies the parameter you are interrogating. This parameter is required.

integer1

Number describing the position of the value set for the parameter in question. The default value is 1.

Use this parameter if the parameter is defined as multiple value sets, each set having one value.

integer2

Number describing the position of the value element for the parameter in question. The default value is 1.

Use this parameter if the parameter is defined as multiple value sets, each set having multiple value elements.

Remarks

- This function is used to reference parameters within a procedure.
- If this function returns a TRUE value, the parameter was given as a range. If this function returns a FALSE value, the parameter was not given as a range.
- For further information about functions, see the NOS/VE System Usage manual.

Examples The examples are based on the following procedure header:

```
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
  )
```

- Consider the following call to the preceding procedure:

  ```
  /display_number (2,(1..10),3)
  ```

  In this case, the second value set is tested for a range by including the following command in the DISPLAY_NUMBER procedure:

  ```
  display_value $range(number,2)
  ```

  The DISPLAY_VALUE command writes TRUE to the output file.

- The next example is based on the following call to procedure DISPLAY_NUMBER:

  ```
  display_number (1,(12..14,16),6)
  ```

  The first and second elements of the second value set are tested for a range by having the following commands in the procedure:

`display_value $range(number,2,1)`	TRUE is written to the output file.
`display_value $range(number,2,2)`	FALSE is written to the output file.

RECOVER_KEYED_FILE Command

Purpose Begins a keyed-file recovery attempt.

Format **RECOVER_KEYED_FILE** or
RECKF
FILE=file
PASSWORD=name
STATUS=status variable

Parameters

FILE or **F**

File path to the damaged keyed file to be recovered. This parameter is required.

If the damaged file does not currently exist, its cycle number cannot be determined by default. Therefore, the file path must explicitly specify the file cycle number so that the utility can reload the correct backup copy.

PASSWORD or *PW*

File password specified when Backup_Permanent_File wrote the backup copy of the file. A file password is optional, but, if a password exists for the file, it is required on this command. If no password exists for the file, NONE can be specified.

The file password in effect when the backup copy was written must be the same password in effect when the file was damaged. Otherwise, the backup copy cannot replace the damaged file.

Remarks

- The LOG_RESIDENCE attribute of the file specified on the command must match the LOG_RESIDENCE attribute of the backup copy to be reloaded. Recover_Keyed_File cannot use a backup copy that was written before the LOG_RESIDENCE attribute of the file was changed.

- If the file does not currently exist and the LOG_RESIDENCE of its backup copy is not the default log, you must enter a SET_FILE_ATTRIBUTE command for the file. The command must specify the same file cycle specified on the RECOVER_KEYED_FILE command and the same LOG_RESIDENCE as that of the backup copy to be used. (See the Example.)

- Similarly, if the file does not currently exist, but the file had a password when the backup copy was written, you must create the file with the same password. To do so, enter a CREATE_FILE command specifying the file path (including its cycle number) and the PASSWORD parameter.

- For more information, see the NOS/VE Advanced File Management Usage manual.

Examples The following session attempts to restore a keyed file that no longer exists using its latest backup copy. When the latest backup copy was written, the file password was HUSH_HUSH and the LOG_RESIDENCE attribute was $USER.MY_LOG. Therefore, those values must be reestablished for the file cycle.

```
/recover_keyed_file, $user.keyed_file.1
reckf/create_file, $user.keyed_file.1, ..
reckf../password=hush_hush
reckf/set_file_attribute, $user.keyed_file.1, ..
reckf../log_residence=$user.my_log
reckf/recover_file_media
```

RELEASE_RESOURCE Command

Purpose Releases tape reservations previously established with the RESERVE_RESOURCE command.

Format

RELEASE_RESOURCE or
RELEASE_RESOURCES or
RELR
MT9$800=integer or *keyword*
MT9$1600=integer or *keyword*
MT9$6250=integer or *keyword*
STATUS=status variable

Parameters *MT9$800*

Specifies the integer number of 9-track tapes with 800-cpi density that are no longer required by the job. If ALL is specified, all of the unused resources of the class and density defined by the parameter name are released. Omission causes 0 to be used.

MT9$1600

Specifies the number of 9-track tapes with 1600-cpi density that are no longer required by the job. If ALL is specified, all of the unused resources of the class and density defined by the parameter name are released. Omission causes 0 to be used.

MT9$6250

Specifies the number of 9-track tapes with 6250-cpi density that are no longer required by the job. If ALL is specified, all of the unused resources of the class and density defined by the parameter name are released. Omission causes 0 to be used.

Remarks For more information, see the NOS/VE System Usage manual.

Examples 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
```

$REMOTE_VALIDATION Function

Purpose Returns a boolean value indicating whether you are validated for file access at the specified remote location.

Format

```
$REMOTE_VALIDATION
   (name)
```

Parameters

name

Specifies the name of the remote location at which the files to be accessed reside. If validation is established for this location, the function returns a TRUE value. This parameter is required.

Remarks

- For details about how to establish remote validation, see the CREATE_REMOTE_VALIDATION command.
- For further information about functions, see the NOS/VE System Usage manual.

Examples The following example queries whether remote validation is established for the named location (MACHINE_A) and, if it is not established, displays a message:

```
IF $remote_validation(machine_a) = FALSE THEN
  display_value 'remote validation not defined ..
  for MACHINE_A'
IFEND
```

REPEAT
Control Statement

Purpose Provides for conditional repetition of a statement list.

Format

label: **REPEAT**
statement list
UNTIL boolean expression

Parameters

label

Specifies the name of the REPEAT block. This label can be used by CYCLE or EXIT statements within the block.

statement list

Specifies the statements that reside in the block.

boolean expression

Specifies the terminating condition of the REPEAT statement. This parameter is required.

Remarks

- An ending label is not allowed for the REPEAT statement.
- For more information, see the NOS/VE System Usage manual.

Examples 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.

REPLACE_FILE Command

Purpose Transfers a copy of a NOS/VE file to a NOS direct or indirect access permanent file or to a NOS/BE file.

Format

REPLACE_FILE or
REPF
FROM=file
TO=name
DATA_CONVERSION=keyword
USER=name
PASSWORD=name
EXCLUSIVE_ACCESS=name
CYCLE=integer
STATUS=status variable

Parameters

FROM or **F**

Identifies the NOS/VE file to be copied into a NOS or NOS/BE file and, optionally, specifies how the file is to be positioned prior to use. This parameter is required.

TO or *T*

Specifies the name of the NOS or NOS/BE file to be replaced or created.

In NOS, this is the permanent file name as registered in the NOS file system and can be up to 7 characters in length. Omission causes the permanent file name in the FROM parameter to be used.

In NOS/BE, if the TO parameter is omitted, the file specified on the FROM parameter is used and the file name part of the FROM parameter must conform to the NOS/BE permanent file naming conventions (except that the length will be limited to 81 characters).

DATA_CONVERSION or *DC*

Specifies the type of conversion to be done during the file copy. The possible keywords are:

B60

The rightmost 60 bits of each 64-bit NOS/VE word are placed into a 60-bit NOS or NOS/BE word. The leftmost 4 bits of each NOS/VE word are ignored.

B56

Contiguous bits from the NOS/VE words are packed into the rightmost 56 bits of each NOS or NOS/BE word. The leftmost 4 bits of NOS words are set to 0 (zero). The leftmost 4 bits of each NOS/BE word are ignored. Specify B56 for saving files such as NOS/VE object libraries, SCU libraries, or permanent file backup files.

A6

Each 7-bit ASCII character (right-justified in an 8-bit byte) is converted to NOS 6/12 display code representation in the NOS or NOS/BE file.

A8

Each 7-bit ASCII character (right-justified in an 8-bit byte) is converted to 12-bit ASCII code format in the NOS or NOS/BE file.

D63

Each 7-bit ASCII character (right-justified in an 8-bit byte) is converted to 6-bit display code (63-character subset of the ASCII 128-character set) format in the NOS or NOS/BE file.

D64

Each 7-bit ASCII character (right-justified in an 8-bit byte) is converted to 6-bit display code (64-character subset of the ASCII 128-character set) format in the NOS or NOS/BE file.

A63, B32, and B64 values are not supported. Omission of this parameter causes A6 to be used.

USER or *U* or *ID*

Specifies the NOS user identification of the owner of the file.

In NOS, this parameter is only necessary if the file is registered in a catalog belonging to a user whose identification is different from your NOS identification (specified on a prior CHANGE_LINK_ATTRIBUTE command).

In NOS/BE, this parameter is not required if the NOS/BE file id is the same as the current user name or the name specified in the last CHANGE_LINK_ATTRIBUTE command.

PASSWORD or *PW* or *TURNKEY* or *TK*

In NOS, this parameter specifies the NOS file password needed to access the file. It is only required when the file does not belong to you.

In NOS/BE, this parameter specifies NOS/BE permanent file permissions. It is required only if you wish to deny all access to a file without turnkey password specification. If you specify any passwords when a NOS/BE file is initially created, you must also specify these passwords on any subsequent REPLACE_FILE commands for the file.

EXCLUSIVE_ACCESS or *XR*

This parameter applies to NOS/BE only and is required if you wish to limit users to read access without password specification.

CYCLE or *CY* or *C*

This parameter specifies a NOS/BE file cycle number and is only applicable to NOS/BE. It is required only if a specific cycle of the file is to be replaced.

Remarks

- If a NOS permanent file of the same name already exists and you have write access to the file, then its contents are replaced by a copy of the NOS/VE file.
- If no NOS direct or indirect permanent file of the specified name exists in the catalog, an attempt to create a direct access file is made. If you are not validated to create direct access files, an indirect access file is created.
- A CHANGE_LINK_ATTRIBUTE command issued prior to the REPLACE_FILE identifies the accounting and user identification information needed to access the file.

- REPLACE_FILE does not preserve the attributes of the NOS/VE file. Therefore, a subsequent GET_FILE command may not create the file with the proper attributes. For information on how to preserve NOS/VE file attributes, see the SCL System Interface manual.
- If the data conversion is D64, A6, or A8, the NOS or NOS/BE file will be written with zero-byte terminated (Z-type) records.
- In NOS/BE, when you enter a REPLACE_FILE command the system copies the NOS/VE file to the NOS/BE default permanent file set and generates the NOS/BE CATALOG command to make the file permanent.
- For more information, see the NOS/VE System Usage manual.

Examples NOS

The following command copies NOS/VE file NEW_PROLOG to NOS file NPROLOG, which is stored under the current user name.

```
/repf from=new_prolog to=nprolog
```

NOS/BE

The following example copies NOS/VE file DATAFIL from the master catalog to NOS/BE file DATAFIL with a file id of RJG and an EXCLUSIVE_ACCESS password of XYZ.

```
/repf from=$user.datafil user=rjg exclusive_access=xyz
```

REPLACE_MULTI_RECORD_FILE Command

Remarks Reserved for site personnel, Control Data, or future use.

REQUEST_LINK Command

Purpose Defines a link file.

Format

REQUEST_LINK or
REQL
FILE=file
STATUS=status variable

Parameters

FILE or **F**

Local file name. This file must not be assigned to another device class; if it is, the command returns an error status and terminates. This parameter is required.

Remarks For more information, see the CYBIL Language Definition manual.

REQUEST_MAGNETIC_TAPE Command

Purpose Associates a file with a tape unit.

Format

REQUEST_MAGNETIC_TAPE or
REQMT
FILE=file
EXTERNAL_VSN=list of string
RECORDED_VSN=list of string
TYPE=keyword
RING=boolean
STATUS=status variable

Parameters

FILE or **F**

Specifies the file to be associated with a magnetic tape unit. This parameter must specify a temporary file. This parameter is required.

EXTERNAL_VSN or *EVSN*

Specifies the identity of one or more tape volumes to be associated with the file. An external volume serial number (VSN) is used to inform the operator which tape is to be mounted. If more than one external VSN is specified, the tapes are requested in the order specified in this list.

Specify a string of 1 to 6 alphanumeric characters. An external VSN of less than 6 characters is left-justified with trailing spaces added. Omission causes a string of spaces to be used.

If the EXTERNAL_VSN parameter is omitted but the RECORDED_VSN parameter is specified, the RECORDED_VSN informs the operator which volume is to be mounted.

It is recommended that the external VSN be visible on the canister containing the volume to be mounted.

RECORDED_VSN or *RVSN*

Specifies the VSN corresponding to the volume identifier recorded in an ANSI VOL1 label. If you enter this parameter for an unlabelled tape, it is ignored.

If the RECORDED_VSN parameter is omitted for a labelled tape, NOS/VE uses the EXTERNAL_VSN parameter to verify that the correct volume as identified by the VOL1 label has been mounted.

If both the RECORDED_VSN and the EXTERNAL_VSN parameters are specified, NOS/VE uses the EXTERNAL_VSN parameter to direct the system operator.

Specify 1 to 6 characters from any of these groups:

Integers 0 to 9.
Uppercase letters A to Z.
These characters:

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

Where SP represents a blank.

An external VSN of less than 6 characters is left-justified with trailing spaces added. Omission causes a string of spaces to be used.

Lists of external and recorded VSN's can be specified for the tape file. If both the external VSN list and the recorded VSN list are specified the corresponding entries in the lists are paired.

If the number of VSN's in the external and recorded VSN lists are different, the system rejects the REQUEST_MAGNETIC_TAPE command.

If both the external VSN and the recorded VSN lists are omitted, the system requests the operator to mount a scratch tape. If a scratch tape is requested for a labelled

tape file, NOS/VE uses the EXTERNAL_VSN assigned by the operator to verify that the correct volume identified by the VOL1 label has been mounted.

TYPE or *T*

Specifies the type of tape transport required. The values are:

MT9$800

Nine-track magnetic tape, 800-cpi density.

MT9$1600

Nine-track magnetic tape, 1600-cpi density.

MT9$6250

Nine-track magnetic tape, 6250-cpi density.

Omission causes MT9$1600 to be used.

RING or *R*

Specifies if a write ring should be present in each volume mounted for this file. Omission causes FALSE to be used (the tape cannot be written).

Remarks

- 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 that is currently associated with a different device class, such as disk or terminal, an error status is returned.
- A write ring is required to write on a tape.
- If you issue a request for a tape file that spans more than one volume, you must list all relevant VSNs on the EXTERNAL_VSN parameter.
- For more information, see the NOS/VE System Usage manual.

Examples The following example assigns a nine-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 assigns a nine-track, 6250-cpi multivolume tape file with external VSNs Y4567, Y4568, and Y4569 to file NEWPL.

```
/request_magnetic_tape newpl external_vsn=('Y4567', ..
../'Y4568','Y4569') type=mt9$6250
```

REQUEST_OPERATOR_ACTION Command

Purpose Sends a message to the system operator and requests a reply message from the operator.

Format

REQUEST_OPERATOR_ACTION or
REQOA
MESSAGE = string
REPLY = string variable
STATUS = status variable

Parameters

MESSAGE or **M**

Specifies the message string to be displayed to the operator. The job is suspended until the operator issues a REPLY_ACTION command in response to this action request. This parameter is required.

REPLY or *R*

Specifies an SCL string variable in which the reply message from the operator's REPLY_ACTION command is placed. Omission causes the reply message to be put into the requesting job's log and also to be written to the $RESPONSE file.

Remarks

- The job is suspended while it is waiting for the operator to respond.
- For more information, see the NOS/VE System Usage manual.

Examples The following example sends a message to the system operator.

```
/reply_string=''
/request_operator_action ..
../message='Do you have tape canister PF001?' ..
../reply=reply_string
```

The job is suspended until the operator responds to your message. For example:

```
/display_value reply_string
SORRY, COULD NOT LOCATE PF001
```

REQUEST_TERMINAL Command

Purpose Associates a file with a terminal in an interactive job.

Format

REQUEST_TERMINAL or
REQT
FILE = file
ATTENTION_CHARACTER_ACTION = integer
BREAK_KEY_ACTION = integer
END_OF_INFORMATION = string
INPUT_BLOCK_SIZE = integer
INPUT_EDITING_MODE = keyword
INPUT_OUTPUT_MODE = keyword
INPUT_TIMEOUT = boolean
INPUT_TIMEOUT_LENGTH = integer
INPUT_TIMEOUT_PURGE = boolean
PARTIAL_CHARACTER_FORWARDING = boolean
PROMPT_FILE = file
PROMPT_STRING = string
STORE_BACKSPACE_CHARACTER = boolean
STORE_NULS_DELS = boolean
TRANSPARENT_CHARACTER_MODE = keyword
TRANSPARENT_FORWARD_CHARACTER = list of string
TRANSPARENT_LENGTH_MODE = keyword
TRANSPARENT_MESSAGE_LENGTH = integer
TRANSPARENT_TERMINATE_CHARACTER = list of string
TRANSPARENT_TIMEOUT_MODE = keyword
STATUS = status variable

Parameters

FILE or F

Specifies the file to be created and associated with your terminal. This parameter can specify only a local file. This parameter is required.

ATTENTION_CHARACTER_ACTION or *ACA*

Specifies how the network responds when it recognizes an ATTENTION_CHARACTER in the data from your terminal. Values can be any integer from 0 to 9.

BREAK_KEY_ACTION or *BKA*

Specifies how the network responds when it recognizes a BREAK signal from you terminal. Values can be any integer from 0 to 9.

END_OF_INFORMATION or *EOI*

Specifies the string (0 to 31 characters) that marks end-of-information in a file.

INPUT_BLOCK_SIZE or *IBS*

Specifies the maximum number of characters the network can hold before forwarding the data. Values can be any integer from 80 to 2000.

INPUT_EDITING_MODE or *IEM*

Specifies whether the network edits the data you enter at your terminal. Values are NORMAL (which enables editing) or TRANSPARENT (which tells the system not to perform editing).

INPUT_OUTPUT_MODE or *IOM*

Specifies whether the network gives input priority over output. Values can be:

UNSOLICITED (U)

Indicates that input has priority over output. The network edits and forwards input as soon as it is received. This action delays any output that may be in progress.

SOLICITED (S)

Indicates that the service must request input. The network does not edit or forward input until requested.

FULL_DUPLEX (F)

Indicates that the network edits and forwards input as soon as it is received. The network also receives and forwards output whenever NOS/VE sends it.

INPUT_TIMEOUT or *IT*

Specifies whether you want input timeout to be in effect. Enter TRUE or FALSE. If you specify TRUE, input timeout is enabled.

INPUT_TIMEOUT_LENGTH or *ITL*

Specifies the number of milliseconds (0 to 86,401) the network will wait for input before timing out. If you specify 0 (zero), the system returns an error condition indicating that no data is available.

INPUT_TIMEOUT_PURGE or *ITP*

Specifies whether the network should purge the input and output paths if an input timeout occurs. If you specify TRUE, the input and output paths will be purged.

If input timeout is enabled and the input timeout length is nonzero, the system returns an input timeout error condition if the time limit is exceeded. Also, the system performs an input timeout purge operation.

There is no effect if you specify 0 (zero) milliseconds.

PARTIAL_CHARACTER_FORWARDING or *PCF*

Specifies whether the network forwards a partial message when an END_PARTIAL_CHARACTER occurs. Enter TRUE or FALSE. Specifying TRUE means that partial character forwarding is enabled.

PROMPT_FILE or *PF*

Specifies the file to which the system should write the automatically generated prompt string. If omitted, the existing parameter value applies. The default is $OUTPUT.

PROMPT_STRING or *PS*

Specifies the string (0 to 31 characters) that the system will output when a program requests terminal input. The system assumes the string contains a format effector. If omitted, the existing parameter value applies. The default is a question mark (?).

STORE_BACKSPACE_CHARACTER or *SBC*

Specifies how the network handles the backspace character. If you specify TRUE, the network forwards the backspace character as part of NORMAL mode data. If you specify FALSE, the network discards the backspace character from NORMAL mode data after it deletes the previous character.

STORE_NULS_DELS or *SND*

Specifies how the network handles the NUL and DEL characters. If you specify TRUE, the network forwards the NUL and DEL characters to the service as part of NORMAL mode data. If you specify FALSE, the network discards the NUL and DEL characters from NORMAL mode data.

TRANSPARENT_CHARACTER_MODE or *TCM*

Identifies the action the network takes when data entered at your terminal contains a TRANSPARENT_FORWARD_CHARACTER or TRANSPARENT_TERMINATE_CHARACTER. The network processes these characters only if INPUT_EDITING_MODE is set to TRANSPARENT. Values can be:

TERMINATE (T)

Sends your data to the service and terminates TRANSPARENT mode when TRANSPARENT_TERMINATE_CHARACTER occurs in input.

FORWARD (F)

Sends your data to the service when a TRANSPARENT_FORWARD_CHARACTER occurs in input. TRANSPARENT mode remains in effect.

FORWARD_TERMINATE (FT)

Sends your data to the service when a TRANSPARENT_FORWARD_CHARACTER occurs in input. This parameter ends TRANSPARENT mode when a TRANSPARENT_TERMINATE_CHARACTER occurs after a TRANSPARENT_FORWARD_CHARACTER in input.

NONE (N)

Takes no action when a TRANSPARENT_FORWARD_CHARACTER or TRANSPARENT_TERMINATE_CHARACTER is received from the terminal.

TRANSPARENT_FORWARD_CHARACTER or *TFC*

Identifies the key you press to forward information. The network only recognizes this attribute if INPUT_EDITING_MODE is set to TRANSPARENT. Values can be a list of 1 to 4 character strings.

TRANSPARENT_LENGTH_MODE or *TLM*

Identifies the action the network takes when it has received the number of characters specified by TRANSPARENT_MESSAGE_LENGTH. The network only performs this action when INPUT_EDITING_MODE is set to TRANSPARENT. Values can be:

TERMINATE (T)

Sends your data to the service and terminates TRANSPARENT mode when the TRANSPARENT_MESSAGE_LENGTH is reached in input.

FORWARD (F)

Sends your data to the service when the TRANSPARENT_MESSAGE_LENGTH is reached in input. The message may exceed the specified length.

FORWARD_EXACT (FE)

Sends the exact data length to the service when the TRANSPARENT_MESSAGE_LENGTH is reached in input.

NONE (N)

Takes no action when the TRANSPARENT_MESSAGE_LENGTH is reached.

TRANSPARENT_MESSAGE_LENGTH or *TML*

Specifies the number of characters in data. The network only recognizes this parameter when INPUT_EDITING_MODE is set to TRANSPARENT. Values can be any integer from 1 to 32,767.

TRANSPARENT_TERMINATE_CHARACTER or *TTC*

Identifies the key you press to terminate and forward input. The network only recognizes this parameter when INPUT_EDITING_MODE is set to TRANSPARENT. Values can be a list of 1 to 4 character strings.

TRANSPARENT_TIMEOUT_MODE or *TTM*

Identifies the action the network takes after a period of inactivity during input (timeout). Values can be one of:

TERMINATE (T)

Terminates TRANSPARENT mode when a timeout of 400 milliseconds or more occurs in input.

FORWARD (F)

Sends your data to NOS/VE when a timeout of 400 milliseconds or more occurs between characters.

NONE (N)

Takes no action when timeout occurs between characters.

Remarks

- If this command is issued for a file that is already assigned to a different device class, such as magnetic tape or disk, an error status is returned.
- Note that your terminal will become inoperable if all three of the following conditions are met:

  1. The INPUT_EDITING_MODE connection attribute has been specified as TRANSPARENT.
  2. The TRANSPARENT_CHARACTER_MODE, TRANSPARENT_TIMEOUT_MODE, and TRANSPARENT_LENGTH_MODE attributes have not been specified.

3. Input occurs while output is being written to the connected terminal file.

   To correct this problem, be sure to specify a value other than NONE on the TRANSPARENT_CHARACTER_MODE, TRANSPARENT_TIMEOUT_MODE, and TRANSPARENT_LENGTH_MODE parameters of this command.

- For more information, see the NOS/VE System Usage manual.

Examples The following command associates local file ALTERNATE_INPUT_FILE with a terminal.

```
/request_terminal file=alternate_input_file
```

A program that subsequently issues a read request to this file will instead issue a read request to your terminal.

In the following example, file ALTERNATE_INPUT_FILE is assigned to the terminal. A string variable name STRING1 is created and a subsequent ACCEPT_LINE command reads file ALTERNATE_INPUT_FILE for the value of the string. A PUT_LINE command then writes the string to file OUTPUT, which by default is assigned to the terminal, and the output is displayed on the screen.

```
/request_terminal file=alternate_input_file
/create_variable string1 kind=string
/accept_line string1 input=alternate_input_file ..
../p='SUPPLY STRING1'
SUPPLY STRNG1 -Input to STRING1
/put_line string1

Input to STRING1
```

In this example, the first character of STRING1 (the hyphen) is interpreted as a format effector. The hyphen character causes the system to space down three lines before printing the string.

RESEQUENCE Command

Remarks Reserved for site personnel, Control Data, or future use.

RESUME_COMMAND Command

Purpose Resumes any job activity that was interrupted because of a pause break.

Format

RESUME_COMMAND or
RESC
STATUS=status variable

Remarks

- The state of the job is returned to what existed before the interruption except for any alterations that you might have made.
- This command is valid only while activity is suspended after a pause break.
- For more information, see the NOS/VE System Usage manual.

Examples Assume that you have initiated a command and are unsure of what to do next. The following sequence returns control to you.

```
/set_password
Enter old password
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX      Pause break entered.
*Suspended - 1*
p/
```

The SET_PASSWORD command is suspended by entering a pause break. The system responds '*Suspended - 1*' to inform you that this is pause break 1 (the 'p/' prompt indicates to you that a pause break is in effect). You can now enter any valid command.

For example:

```
p/help set_password
```

could be entered to receive an online explanation of the SET_PASSWORD command.

In the following example, the SET_PASSWORD command is resumed and is expecting entry of the old password (any previous prompts are not reissued after a RESUME_COMMAND).

```
p/resume_command
pass789                                    Old password entered.
Enter new password
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

RESERVE_RESOURCE Command

Purpose Specifies the number of tape units a job requires.

Format

RESERVE_RESOURCE or
RESERVE_RESOURCES or
RESR

MT9$800=integer
MT9$1600=integer
MT9$6250=integer
STATUS=status variable

Parameters

MT9$800

Specifies the number of nine-track tapes with 800-cpi density that are required by the job. Omission causes 0 (zero) to be used.

MT9$1600

Specifies the number of nine-track tapes with 1600-cpi density that are required by the job. Omission causes 0 (zero) to be used.

MT9$6250

Specifies the number of nine-track tapes with 6250-cpi density that are required by the job. Omission causes 0 (zero) to be used.

Remarks

- This information is used for job scheduling to prevent deadlocks with other jobs that may need the same resources.
- Actual equipment assignment is not made until a tape file is opened for access.
- For more information, see the NOS/VE System Usage manual.

Examples The following example reserves three nine-track magnetic tape units: one with 800 cpi and two with 6250 cpi.

```
/reserve_resources mt9$800=1 mt9$6250=2
```

RESTORE_LOG Command

Purpose Begins a Restore_Log utility session.

Format

RESTORE_LOG or
RESL
LOG_RESIDENCE=file
STATUS=status variable

Parameters

LOG_RESIDENCE or **LR**

Catalog path containing the files composing the log to be restored. This parameter is required.

Remarks

- Immediately after entering the Restore_Log session, you should use the VALIDATE_LOG or RESTORE_REPOSITORIES subcommands to determine the type and extent of log damage, if any.
- For more information, see the NOS/VE Advanced File Management Usage manual.

RESTORE_PERMANENT_FILES Command

Purpose Initiates the utility that restores permanent files and catalogs from backup copies created by the BACKUP_PERMANENT_FILE utility. The restore operations are directed by RESTORE_PERMANENT_FILE subcommands.

Format

RESTORE_PERMANENT_FILES or
RESTORE_PERMANENT_FILE or
RESPF
LIST=file
STATUS=status variable

Parameters *LIST* or *L*

Identifies the file to which a summary of the results of the restore utility are written and, optionally, specifies how the file is to be positioned prior to use. Omission causes $LIST to be used.

Remarks

- The content of the list file can be specified using the SET_LIST_OPTION subcommand prior to using a RESTORE_PERMANENT_FILE subcommand. If the SET_LIST_OPTION subcommand is omitted, the modification date and time and size of the file are displayed for each permanent file cycle.
- For more information, see the NOS/VE System Usage manual.

Examples The following subcommand initiates a RESTORE_PERMANENT_FILE subcommand utility session. The subcommand specifies that the report listing be written to file RESTORE_LISTING.

```
/restore_permanent_files list=restore_listing
```

Following entry of this subcommand, RESTORE_PERMANENT_FILE subcommands can be entered in response to the following prompt.

```
PUR/
```

REWIND_FILE Command

Purpose Positions a file to the beginning-of-information.

Format

REWIND_FILE or
REWIND_FILES or
REWF
FILES=list of file
STATUS=status variable

Parameters **FILES** or **FILE** or **F**

Specifies the list of local files to be rewound. This parameter is required.

Remarks

- 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 next ANSI labelled file accessed is the same as the previously accessed file. For FILE_SET_POSITION values other than NEXT_FILE, this command has no effect. (Refer to the CHANGE_TAPE_LABEL_ATTRIBUTES command description of the FILE_SET_POSITION parameter.)
- For more information, see the NOS/VE System Usage manual.

Examples The following example rewinds several files.

```
/rewind_file file=source
/rewind_files file=(library,test,scratch)
```

$RING Function

Purpose Returns an integer indicating the current execution ring for a task.

Format **$RING**

Parameters None.

Remarks For further information about functions, see the NOS/VE System Usage manual.

Examples The following example indicates that the current task is associated with execution ring 11:

```
/display_value $ring
11
```

ROUTE_JOB Command

Purpose Specifies the name by which a job is to be known, the system on which the job is to be executed, and the destination of the output file generated by the job.

NOTE

This command can only be used for card input or for input from a microcomputer or terminal which supports HASP protocol.

Format

ROUTE_JOB or
ROUJ
JOB_NAME=name
JOB_DESTINATION=name
JOB_OUTPUT_DESTINATION=name
USER_NAME=name
USER_FAMILY=name
STATUS=status variable

Parameters

JOB_NAME or *JN*

Specifies the user job name in upper-case, alphanumeric characters. The default value is your user name.

JOB_DESTINATION or *JD*

Specifies the NOS/VE family name to which the job is to be sent. If this parameter is not specified, the input job will be sent to the DEFAULT_JOB_DESTINATION defined for the I/O station. If this job destination is unavailable, the input job will either be discarded, or the input device stopped, according to the DESTINATION_AVAILABLE_ACTION defined for the I/O station.

JOB_OUTPUT_DESTINATION or *JOD*

Specifies a public I/O station or the control facility of a private I/O station, to which the job's output file(s) will be sent. If the specified destination is a control facility, the USER_NAME and USER_FAMILY parameters must also be included to uniquely identify the private I/O station.

If no JOB_OUTPUT_DESTINATION value is specified, output is returned to the I/O station where the job was initiated.

USER_NAME or *UN*

Specifies the user controlling the private I/O station to which the job's output file(s) are to be sent. This parameter must be specified if the value of the JOB_OUPUT_DESTINATION parameter is the control facility for a private I/O station.

If the value for this parameter is not specified, the value of the JOB_OUTPUT_DESTINATION parane ter s assumed to be a public I/O station.

USER_FAMILY or *UF*

Specifies the family in which the user identified by the USER_NAME parameter is validated. This parameter is valid only if the USER_NAME parameter is specified.

Remarks

- The ROUTE_JOB command must precede job input (the LOGIN command) from the I/O station.
- For card input or for microcomputers that emulate HASP protocol, the command must start at column six with the ASCII string '/*BC' (specifying a Batch Command text) in columns one through five. This same format applies to any continuation card(s) needed to specify the command.
- The length of this command cannot exceed 256 characters.
- A ROUTE_JOB command error causes the input job to be discarded and an error message to be sent to the I/O station operator.
- For more information, see the NOS/VE System Usage manual.

Examples

The following example routes JOB1 to system NVE for execution, and sends the job output to public I/O station PUBLIC_STATION_1.

```
/*BC route_job jn=job1 jd=nve jod=public_station_1
login login_user=john password=XXX
collect_text output
This is just a test
**
/*EOI
```

The following example submits JOB2 to system NVE for execution and sends the job output, via the control facility, to the I/O station controlled by :NVE.JOHN.

```
/*BC route_job jn=job2 ..
/*BC jd=nve ..
/*BC jod=NVE_control_facility ..
/*BC un=john ..
/*BC uf=nve
login login_user=john pass_word=xxx
collect_text output
This is just a test
**
/*EOI
```

$SCAN_ANY Function

Purpose Searches a string for any one of a specified set of characters.

Format

$SCAN_ANY
(string1
string2)

Parameters

string1

Specifies the set of characters being searched for. This parameter is required.

string2

Specifies the string being searched. This parameter is required.

Remarks

- This function returns a number indicating the position of the first character in the first string that is also found in the second string. If no character from the first string appears in the second string, the integer 0 is returned.
- For further information about functions, see the NOS/VE System Usage manual.

Examples The following example looks for the position of the first character in string S that also occurs in string D:

```
/d = '0123456789'
/s = 'temp_32'
/display_value $scan_any(d,s)
6
```

The number 6 is returned because the first character in string S that also occurs in string D is the number 3; this number is in the sixth character position within string S.

$SCAN_NOT_ANY Function

Purpose Searches a string for any character that is not in a specified set of characters

Format

$SCAN_NOT_ANY or
$SCAN_NOTANY
(string1
string2)

Parameters

string1

Specifies the string containing the set of characters. The system searches for any character not in this set. This parameter is required.

string2

Specifies the string being searched. This parameter is required.

Remarks

- This function returns an integer indicating the position of the first character in the second string that is not in the first string. If only characters from the first string appear in the second string, the integer 0 is returned.
- For further information about functions, see the NOS/VE System Usage manual.

Examples The following example returns an integer showing the position of the first character in string S (namely, *t*) that is not found in string D:

```
/d = /0123456789'
/s = 'temp_32'
/display_value $scan_notany(d,s)
1
```

$SCAN_STRING Function

Purpose Searches a string to locate occurrences of another string (called the pattern).

Format

$SCAN_STRING
(string1
string2)

Parameters

string1

Specifies the pattern string (string being searched for). This parameter is required.

string2

Specifies the string being searched. This parameter is required.

Remarks

- This function returns an integer indicating the position of the first character of the first occurrence of the pattern string in the string being searched. If the pattern is a null string, the integer 1 is returned. If the pattern is not found in the string, the integer 0 is returned.
- For further information about functions, see the NOS/VE System Usage manual.

Examples The following example returns a number showing the position in string S of the first occurrence of pattern string P (namely, the sixth position):

```
/s = '0123_abc9'
/p = 'abc'
/display_value $scan_string(p,s)
6
```

SEARCHPCD
IM/DM Command

Remarks Reserved for site personnel, Control Data, or future use.

SELECT_USER_MENU
Command

Purpose Starts the NOS/VE user menu system.

Format

SELECT_USER_MENU or
SELUM or
MENU
PROLOG_CALL=boolean
NATURAL_LANGUAGE=name
STATUS=status variable

Parameters *PROLOG_CALL* or *PC*

Reserved for use by site administrators.

NATURAL_LANGUAGE or *NL*

Specifies the natural language of the menus and help messages. Omission causes US_ENGLISH to be used. US_ENGLISH is currently the only supported natural language for the NOS/VE menu system.

Remarks For more information, see the NOS/VE System Usage manual.

SET_COMMAND_LIST
Command

Purpose Changes the current command list by deleting and/or adding command list entries, and/or altering the state of the search mode indicator.

NOTE

The preferred commands are now CREATE_COMMAND_LIST_ENTRY, DELETE_COMMAND_LIST_ENTRY, and CHANGE_COMMAND_SEARCH_MODE.

Format

SET_COMMAND_LIST or
SETCL
DELETE=list of file or *keyword*
ADD=list of file or *keyword*
SEARCH_MODE=keyword
PLACEMENT=keyword
SYSTEM_COMMAND_LIBRARY=file or *keyword*
STATUS=status variable

Parameters

DELETE or *D*

Specifies entries to be removed from the current command list. If this parameter is specified as ALL, then all entries in the current command list are deleted. All deletions are performed prior to any additions. Omission causes no entries to be deleted from the current command list.

ADD or *A*

Specifies entries to be added to the front of the current command list. They appear in the command list in the order specified. Omission causes no additions to the current command list.

SEARCH_MODE or *SM*

Specifies the new search mode to be associated with the current command list:

GLOBAL (G)

All entries in the command list can be searched. Commands specified by path name and command name can be executed.

RESTRICTED (R)

All entries in the command list can be searched. However, for a search to proceed beyond the first entry in the command list, the command must be preceded by a slash (/). Commands specified by path name and command name can be executed.

EXCLUSIVE (E)

Only the entry at the beginning of the command list is searched for a command.Commands that are specified by path name and command name are not allowed.

Omission leaves the current search mode unchanged.

PLACEMENT or *P*

Specifies whether the entries added to the command list are placed before or after the current entries. Use one of the following keywords:

AFTER (A)

Causes the entries to be placed after the current entries.

BEFORE (B)

Causes the entries to be placed before the current entries. This is the default.

If the SEARCH_MODE parameter is set to RESTRICTED, the only allowable value for the PLACEMENT parameter is AFTER.

SYSTEM_COMMAND_LIBRARY or *SCL*

Reserved.

Remarks For more information, see the NOS/VE System Usage manual.

Examples An object library named PROCEDURE_LIBRARY can be added to the end of the command list with the following command.

```
/set_command_list add=$user.procedure_library p=after
```

To delete the object library added in the preceding example, enter the following command.

```
/set_command_list delete=procedure_library
```

If you wish to move the $SYSTEM entry to the front of the command list, enter the following command.

```
/set_command_list delete=$system add=$system
```

With the $SYSTEM entry at the front of the command list, system commands take precedence over local files with the same name.

$SET_COUNT Function

Purpose Returns an integer count of the number of value sets actually passed for a specified parameter.

Format

$SET_COUNT
(name)

Parameters

name

Specifies the parameter you are interrogating. This parameter is required.

Remarks

- This function is used to reference parameters within procedures.
- For further information about functions, see the NOS/VE System Usage manual.

Examples The example is based on the following procedure header:

```
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
  )
```

Consider the following call to the preceding procedure:

```
/display_number number = (2,4,5)
```

In this case, the number of value sets for the NUMBER parameter is displayed by including the following command in the DISPLAY_NUMBER procedure:

```
/display_value $set_count(number)
```

The value 3 is written to the output file.

SET_DEBUG_LIST Command

Purpose Adds or deletes debug object libraries from the job debug library list. (See the NOS/VE Object Code Management Usage manual for a description of the job debug library list.)

Format

SET_DEBUG_LIST or
SETDL
DELETE_LIBRARIES=list of file or *keyword*
ADD_LIBRARIES=list of file
STATUS=status variable

Parameters

DELETE_LIBRARIES or *DELETE_LIBRARY* or *DL*

Specifies the debug object libraries to be deleted from the job debug library list. Options are:

Omitted

No libraries are deleted from the job debug library list.

list of file

Deletes the specified library or libraries from the job debug library list.

ALL

All entries in the job debug library list are deleted.

ADD_LIBRARIES or *ADD_LIBRARY* or *AL*

Specifies the debug object libraries to be added to the job debug library list. Options are:

Omitted

No libraries are added to the job debug library list.

list of file

Adds the specified library or libraries to the job debug library list. The libraries are added, in the order specified, at the beginning of the job debug library list.

Remarks

- The job debug library list is the set of debug object libraries used for debugging programs.
- The job debug library list initially contains the system-supplied Debug utility. With SET_DEBUG_LIST, you can specify a user-written debugger to be available in your job.

- The job debug library list is added to the program library list (see the NOS/VE Object Code Management Usage manual for a description or the program library list) whenever the debugger is required, that is, when the program attribute DEBUG_MODE is ON.
- If the program attribute DEBUG_MODE is ON, the loader gives control of program execution to the debugger. Otherwise, the loader initiates program execution by calling the starting procedure.
- An object library is recognized as a debug object library by having a FILE_CONTENTS attribute of OBJECT, a FILE_STRUCTURE attribute of LIBRARY, and a FILE_PROCESSOR attribute of DEBUGGER.
- The DELETE_LIBRARY parameter is always processed before the ADD_LIBRARY parameter and the same library can be specified on both parameters on the same command. This allows you to reorder the job debug library list with a single command.

Examples In the following example, CID_180_LIBRARY is a debug object library added to the job debug library list. Because the program attribute DEBUG_MODE is ON, the loader gives control of program execution to the debugger specified in the job debug library list, that is, CID_180_LIBRARY.

```
/set_debug_list add_library=cid_180_library
/set_program_attributes debug_mode=on
/execute_task lgo
```

SET_DEBUG_RING Command

Purpose Specifies the ring in which Debug (or the debugger specified on the job debug library list) is to execute.

Format

SET_DEBUG_RING or
SETDR
RING=integer
STATUS=status variable

Parameters **RING or R**

Specifies the Debug ring number. Value must be an integer in the range 1 to 13.

Remarks

- The Debug ring cannot be set to a ring more privileged than the lowest ring for which you are validated.
- The initial setting of the Debug ring is your initial ring of execution which is established by your user name's NOMINAL_RING validation.

Examples The following example changes the Debug ring to 11.

```
/set_debug_ring ring_number=11
```

SET_DM_RELEASE IM/DM Command

Remarks Reserved for site personnel, Control Data, or future use.

SET_FILE_ATTRIBUTES Command

Purpose Establishes the attributes of a file that are used to manage its content and processing.

NOTE

Most attributes have a default value that is used if you do not specify the attribute on the SET_FILE_ATTRIBUTES command. However, the default value is sometimes inappropriate for keyed files. It is therefore recommended that you explicitly specify a value for all relevant keyed-file attributes.

NOTE

Although the following parameters are currently supported for this command, it is recommended that you use the ATTACH_FILE command to specify the values for these parameters.

Format

SET_FILE_ATTRIBUTES or
SET_FILE_ATTRIBUTE or
SETFA
FILE=file
ACCESS_MODES=list of keyword
AVERAGE_RECORD_LENGTH=integer
BLOCK_TYPE=keyword
CHARACTER_CONVERSION=boolean
COLLATE_TABLE_NAME=name or *keyword*
COMPRESSION_PROCEDURE_NAME=list of any or *keyword*
DATA_PADDING=integer
DYNAMIC_HOME_BLOCK_SPACE=boolean
EMBEDDED_KEY=boolean
ERROR_EXIT_PROCEDURE_NAME=name or *keyword*
ERROR_LIMIT=integer
ESTIMATED_RECORD_COUNT=integer
FILE_ACCESS_PROCEDURE_NAME=name or *keyword*
FILE_CONTENTS=name or *keyword*
FILE_LABEL_TYPE=keyword
FILE_LIMIT=integer
FILE_ORGANIZATION=keyword
FILE_PROCESSOR=name or *keyword*
FILE_STRUCTURE=name or *keyword*
FORCED_WRITE=boolean or *keyword*
HASHING_PROCEDURE_NAME=list of any or *keyword*
INDEX_LEVELS=integer
INITIAL_HOME_BLOCK_COUNT=integer
INDEX_PADDING=integer
INTERNAL_CODE=keyword
KEY_LENGTH=integer
KEY_POSITION=integer
KEY_TYPE=keyword
LINE_NUMBER=list of integer
LOADING_FACTOR=integer
LOCK_EXPIRATION_TIME=integer
LOGGING_OPTIONS=list of keyword
LOG_RESIDENCE=file or *keyword*
MAXIMUM_BLOCK_LENGTH=integer
MAXIMUM_RECORD_LENGTH=integer
MESSAGE_CONTROL=list of keyword
MINIMUM_BLOCK_LENGTH=integer

MINIMUM_RECORD_LENGTH=integer
OPEN_POSITION=keyword
PADDING_CHARACTER=string
PAGE_FORMAT=keyword
PAGE_LENGTH=integer
PAGE_WIDTH=integer
PRESET_VALUE=integer
RECORD_LIMIT=integer
RECORD_TYPE=keyword
RECORDS_PER_BLOCK=integer
STATEMENT_IDENTIFIER=list of integer
USER_INFORMATION=string
STATUS=status variable

Parameters

FILE or F

Specifies the file whose attributes are being defined. This parameter is required.

For compatability with future NOS/VE releases, it is recommended that this command follow the use of any of the explicit file creation commands, specifically CREATE_FILE, REQUEST_MAGNETIC_TAPE, REQUEST_TERMINAL, and REQUEST_LINK.

If you use the local file name defined by a CREATE_FILE or an ATTACH_FILE command as the FILE parameter for this command, the SET_FILE_ATTRIBUTES command must follow the CREATE_FILE or ATTACH_FILE commands.

ACCESS_MODES or *ACCESS_MODE* or *AM*

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 preceeding selections.

If the file is a permanent file, 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.

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 or *ARL*

Specifies your estimate of the length of the average record in a new keyed file. This parameter 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 (online name AFM).

BLOCK_TYPE or *BT*

Specifies the block type. This parameter 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.

If the file is on disk, a block header is recorded on the file. If the file is an unlabeled tape file, writing of the block header to the tape volume is suppressed. Blocks in memory are preceded by the block header followed by MAXIMUM_BLOCK_LENGTH bytes of buffer space. The block header includes a field that defines the actual length of the block.

Blocks are padded with the circumflex (^) character up to the MINIMUM_BLOCK_LENGTH. MAXIMUM_BLOCK_LENGTH for a tape file is constrained to 4,128 bytes or less.

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

CHARACTER_CONVERSION or *CC*

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 or *CTN*

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

For further information, see the SCL Advanced File Management manual (online name AFM).

COMPRESSION_PROCEDURE_NAME or *CPN*

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

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

DATA_PADDING or *DP*

Specifies the percentage of space within each new data block of an indexed-sequential file that is to be left unused left unused during initial file creation. This parameter 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 (online name AFM).

DYNAMIC_HOME_BLOCK_SPACE or *DHBS*

Reserved.

EMBEDDED_KEY or *EK*

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

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

For further information, see the SCL Advanced File Management manual (online name AFM).

ERROR_EXIT_PROCEDURE_NAME or *EEPN* or *ERROR_EXIT_NAME* or *EEN*

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 parameter is equivalent to the ERROR_EXIT_PROCEDURE_NAME parameter and can be used interchangeably.

Omission causes no error exit procedure to be used.

ERROR_LIMIT or *EL*

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

For details, see the SCL Advanced File Management manual (online name AFM).

ESTIMATED_RECORD_COUNT or *ERC*

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

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

For details, see the SCL Advanced File Management manual (online name AFM).

FILE_ACCESS_PROCEDURE_NAME or *FAPN* or *FILE_ACCESS_PROCEDURE* or *FAP*

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 or *FILE_CONTENT* or *FC*

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 control 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 file.

name

Variable specifying a 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 or *FLT*

Specifies the label type for an ANSI-labeled tape file. This parameter is valid only for systems running PSR level 665. These 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 2

Valid parameter options:

LABELED (L)

Specifies an ANSI standard label.

UNLABELED (UL)

Specifies that the tape is unlabeled.

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

FILE_LIMIT or *FL*

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

The maximum file size is 150,000,000 bytes. Your site may change this maximum value via the system attribute MAXIMUM_SEGMENT_LENGTH.

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 or *FO*

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

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

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

FILE_PROCESSOR or *FP*

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

ADA

ADA compiler.

APL

APL compiler.

ASSEMBLER
NOS/VE assembler.

BASIC
BASIC interpreter.

C
C compiler.

COBOL
COBOL compiler.

CYBIL
CYBIL compiler.

DEBUGGER
DEBUG utility.

FORTRAN
FORTRAN compiler.

LISP
LISP compiler.

PASCAL
Pascal compiler.

PL1
PL1 compiler.

PPU_ASSEMBLER
PP assembler.

PROLOG
PROLOG compiler.

SCL
SCL interpreter.

SCU
Source Code Utility.

UNKNOWN

VX

File processor associated with VX/VE.

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

FILE_STRUCTURE or *FS*

Specifies the structure of the file. This parameter 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 or *FW*

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 parameter is ignored for a sequential or byte-addressable file.

For further information, see the SCL Advanced File Management manual (online name AFM).

HASHING_PROCEDURE_NAME or *HPN*

Specifies the optional, user-defined hashing procedure used only for a direct-access file. This parameter 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 (online name AFM).

INDEX_LEVELS or *INDEX_LEVEL* or *IL*

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

For details, see the SCL Advanced File Management manual (online name AFM).

INITIAL_HOME_BLOCK_COUNT or *IHBC*

Specifies the number of home blocks to be created when a new direct-access file is first opened. This parameter 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 (online name AFM).

INDEX_PADDING or *IP*

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 parameter is ignored for direct access, sequential, or byte-addressable files.

For details, see the SCL Advanced File Management manual (online name AFM).

INTERNAL_CODE or *IC*

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 parameter selections are:

A6

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

A8

NOS 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).

D63

NOS 6-bit display code (CDC 63-character set).

D64

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

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

KEY_LENGTH or *KL*

Specifies the length, in bytes, of the primary key for a new keyed file. KEY_LENGTH is a required parameter for a new indexed sequential file. This parameter 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 (online name AFM).

KEY_POSITION or *KP*

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

For details, see the SCL Advanced File Management manual (online name AFM).

KEY_TYPE or *KT*

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

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

For details, see the SCL Advanced File Management manual (online name AFM).

LINE_NUMBER or *LN*

Specifies the length and location of a line number in each record of a file. The parameter 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 or *LF*

Reserved.

LOCK_EXPIRATION_TIME or *LET*

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

For details, see the SCL Advanced File Management manual (online name AFM).

LOGGING_OPTIONS or *LOGGING_OPTION* or *LO*

Enables the use of keyed-file recovery options. Options are:

ALL

All logging options are enabled.

ENABLE_PARCELS (EP)

Reserved for future use.

ENABLE_MEDIA_RECOVERY (EMR)

An update recovery log is to be maintained for the keyed-file.

ENABLE_REQUEST_RECOVERY (ERR)

An automatic close upon task abort removes from the keyed-file any partially completed update operation caused by system failure.

NONE

No logging options are enabled. This is the default.

For more information on logging options, see the SCL Advanced File Management Usage manual, or the CYBIL Keyed-File and Sort/Merge Interface Usage manual.

LOG_RESIDENCE or *LR*

Specifies the catalog path for the keyed-file's update recovery log. The log must be created by the ADMINISTER_RECOVERY_LOG utility described in the SCL Advanced File Management Usage manual.

Log entries are not written for the file unless its LOGGING_OPTIONS attribute is ENABLE_MEDIA_RECOVERY. If the LOGGING_OPTIONS attribute is ENABLE_MEDIA_RECOVERY, the default for this parameter is $SYSTEM.AAM.SHARED_RECOVERY_LOG.

NOTE

Whenever you change the LOG_RESIDENCE of an existing keyed-file to a log other than the default log, you should immediately backup the file or no entries will be logged. If a backup has not been done since the change and the file is damaged, the RECOVER_FILE_MEDIA subcommand on the RECOVER_KEYED_FILE utility cannot execute successfully for the file.

For more information on logging options, see the SCL Advanced File Management manual and the CYBIL Keyed-File and Sort/Merge Interfaces manual.

MAXIMUM_BLOCK_LENGTH or *MAXBL*

Specifies the maximum block length (0 to 65,497) 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 parameter 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.

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

For keyed files, block length is set at creation time and does not change thereafter. If MAXIMUM_BLOCK_LENGTH is specifed at creation time, the actual block length is the nearest higher number in the series:

2048, 4096, 8192, 16384, 32768, 65536

MAXIMUM_RECORD_LENGTH or *MAXRL*

Specifies the maximum length in bytes (1 to 65,497) allowed for a record. This parameter is used only for files with ANSI fixed length (F) records. F type records are padded to this length on output. This parameter is required for a keyed file, regardless of record type.

Omssion for a new file with ANSI F type records causes 256 to be used. For an old file of ANSI F type records or for an old keyed file, the preserved value is always used.

MESSAGE_CONTROL or *MC*

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

This parameter is ignored for a sequential or byte-addressable file. For details, see the SCL Advanced File Management manual (online name AFM).

MINIMUM_BLOCK_LENGTH or *MINBL*

Specifies, in bytes, the minimum block length. This parameter is applicable only for files with user-specified blocking.

For ANSI fixed records with user-specified blocking, 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 parameter is ignored for keyed files and for files with system-specified blocking.

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

MINIMUM_RECORD_LENGTH or *MINRL*

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

For details, see the SCL Advanced File Management manual (online name AFM).

OPEN_POSITION or *OP*

Specifies the positioning to occur before the file is opened. Options are:

$BOI

Position to beginning-of-information.

$ASIS

No positioning.

$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 parameter causes $EOI to be used for the OUTPUT file and $BOI to be used for all other files.

PADDING_CHARACTER or *PC*

Specifies the padding character used to pad short ANSI fixed records to their MAXIMUM_RECORD_LENGTH.

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

PAGE_FORMAT or *PF*

Specifies the frequency and separation of titling in a legible file. This parameter 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 or *PL*

Specifies the number of lines to be written on a printed page. This parameter 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.

Omission for a new file causes 60 to be used. For an old file, the preserved value is always used.

PAGE_WIDTH or *PW*

Specifies the number of characters to be written to a printed line. This parameter 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.

Omission for a new file causes 132 to be used for a nonterminal file and the value of the PAGE_WIDTH terminal attribute to be used for a terminal file. For an old file, the preserved value is always used.

PRESET_VALUE or *PV*

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 or *RL*

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

For details, see the SCL Advanced File Management manual (online name AFM).

RECORD_TYPE or *RT*

Specifies the record type. Options are:

FIXED (F)

ANSI fixed length.

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 an old file, the preserved value is always used.

RECORDS_PER_BLOCK or *RPB*

Specifies an estimate of the number of data records that are contained in each data block of a new keyed file. This parameter 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 or *SI*

This parameter 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 parameter 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.

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 or *UI*

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.

Remarks

- Only attributes that are noted as applicable to files with a sequential file organization apply to tape files.
- The SET_FILE_ATTRIBUTE command can be used to specify the attributes that are preserved with a new file and those that are related to an instance of open (for example, ACCESS_MODE and OPEN_POSITION).
- When you specify a value for the FILE_CONTENTS parameter but not for the FILE_STRUCTURE parameter, NOS/VE selects a value for the FILE_STRUCTURE parameter.
- When you specify a value for the FILE_STRUCTURE parameter but not for the FILE_CONTENTS parameter, NOS/VE selects a value for the FILE_CONTENTS parameter.
- For compatability with future NOS/VE releases, it is recommended that this command follow the use of any of the explicit file creation commands, specifically CREATE_FILE, REQUEST_MAGNETIC_TAPE, REQUEST_TERMINAL, and REQUEST_LINK.
- To prevent serious performance degradation, the FORCED_WRITE attribute should be set to FALSE if the LOGGING_OPTIONS attribute includes ENABLE_MEDIA_RECOVERY.

- For more information, see the NOS/VE System Usage manual.

Examples

The following command establishes READ and EXECUTE access modes for file TEST_FILE.

```
/set_file_attributes test_file ..
../access_modes=(read,execute)
```

The following command establishes attributes for file TEST_FILE_3 and specifies that its file organization is an indexed sequential file.

```
/setfa test_file_3 file_organization=is
```

The following commands create an empty source file that will contain a FORTRAN object file. The example creates the file with a file content of OBJECT and a file processor of FORTRAN.

```
/setfa program_object file_content=object ..
../file_processor=fortran
```

The following command sets the maximum length of file TEST_FILE_4 to 1 million bytes.

```
/setfa test_file_4 file_limit=1000000
```

The following command establishes file attributes for a disk file that is intended for the local site line printer.

```
/setfa printout page_format=burstable page_length=10 ..
../page_width=40
/colt printout
ct? This file is intended for printing on the line
ct? printer. It has a page length of 10 and page width
ct? of 40.
ct? **
/print_file printout
```

SET_LINK_ATTRIBUTES Command

Purpose Establishes information needed to gain access to NOS or NOS/BE permanent files or to execute a NOS or NOS/BE job in a dual-state system.

NOTE

The preferred command is now CHANGE_LINK_ATTRIBUTES.

Format

SET_LINK_ATTRIBUTES or
SET_LINK_ATTRIBUTE or
SETLA
USER=list of name
PASSWORD=name
CHARGE=string
PROJECT=string
STATUS=status variable

Parameters

USER or **U**

In NOS, USER specifies the NOS user name (from 1 through 7 characters), family name (from 1 through 7 characters), and is required.

In NOS/BE, USER specifies the name used to access the system and is the default permanent file id if a file id is not specified on subsequent file transfer commands. Family name is optional (the system automatically changes to NVE any name entered for the family name parameter).

PASSWORD or **PW**

Specifies the NOS or NOS/BE INTERCOM password. For NOS, the password must be the NOS batch password. Do not confuse this password with a NOS/BE permanent file password.

CHARGE or *C*

In NOS, CHARGE specifies the NOS charge number (from 1 through 10 characters) and is a required parameter.

In NOS/BE, CHARGE specifies the NOS/BE charge number and may be required depending on the accounting level implemented on the system at your site.

PROJECT or *P*

In NOS, PROJECT specifies the NOS project number (from 1 through 20 characters) and is a required parameter.

In NOS/BE, PROJECT specifies the NOS/BE project number and may be required depending on the accounting level implemented on the system at your site.

Remarks

- The command parameters provide the accounting and user identification under which you have been validated to execute on the real-state system. Omission of this command causes the system to use the validation information of the current user.
- Your link attributes are used in conjunction with the following:
  - GET_FILE and REPLACE_FILE .
  - The DUAL_STATE_ROUTE_PARAMETERS output attribute.
  - CREATE_INTERSTATE_CONNECTION utility (documented in the NOS or NOS/BE Migration manual)).
  - The Interstate Communication facility (documented in the CYBIL System Interface manual) .
- For more information, see the NOS/VE System Usage manual.

Examples

In the first example, user SDH changes the default accounting and user identification for subsequent NOS file transfers.

```
/set_link_attributes user=(dlh,nve) ..
../password=pass789 charge='46358a' project='693453a473'
```

Subsequent NOS and NOS/VE file transfers will access user name DLH in family name NVE. User DLH has the password PASS789. The charges for the file transfer will be applied to charge number 46358A and project number 693453A473.

In the next example, user DLH uses the SET_LINK_ATTRIBUTE command to change the default accounting and user identification information for subsequent NOS/BE file transfers.

```
/set_link_attributes user=(dlh,nve) password=pass789 ..
../charge='46358a' project='693453a473'
```

Subsequent NOS/BE and NOS/VE file transfers will access user name DLH with password PASS789 (the NVE family name is ignored). The charges for the file transfer will be applied to charge number 46358A and project number 693453A473. If no permanent file id is specified on subsequent file transfer commands, the user name DLH will be used.

SET_MULTIPROCESSING_OPTIONS Command

Purpose Specifies the CPU(s) on which you want to run a job.

Format

SET_MULTIPROCESSING_OPTIONS or
SET_MULTIPROCESSING_OPTION or
SETMO
MULTIPROCESS=keyword
PROCESSORS=list of keyword
STATUS=status variable

Parameters *MULTIPROCESS* or *MP* or *M*

Specifies whether you want any subsequent jobs to use both CPUs. Values can be ON or OFF. The default is OFF.

PROCESSORS or *PROCESSOR* or *P*

Specifies the processor you want to run on. Choose from P0 or P1.

Remarks For more information, see the NOS/VE System Usage manual.

Examples The following example runs a job on P0 and/or P1.

```
/setmo mp=on
```

The following example runs a job on P1 only.

```
/setmo mp=off p=p1
```

The following example runs a job on P0 or P1, but not both.

```
/setmo mp=off p=(p0,p1)
```

SET_PROGRAM_ATTRIBUTES Command

Purpose Changes default attribute values for subsequent programs executed within the job.

Format

SET_PROGRAM_ATTRIBUTES or
SET_PROGRAM_ATTRIBUTE or
SETPA

LOAD_MAP=file
LOAD_MAP_OPTIONS=list of keyword
PRESET_VALUE=keyword
TERMINATION_ERROR_LEVEL=keyword
DEBUG_INPUT=file
DEBUG_OUTPUT=file
ABORT_FILE=file
DEBUG_MODE=boolean
DELETE_LIBRARIES=list of file or *keyword*
ADD_LIBRARIES=list of file or *keyword*
ARITHMETIC_OVERFLOW=boolean
ARITHMETIC_LOSS_OF_SIGNIFICANCE=boolean
DIVIDE_FAULT=boolean
EXPONENT_OVERFLOW=boolean
EXPONENT_UNDERFLOW=boolean
FP_INDEFINITE=boolean
FP_LOSS_OF_SIGNIFICANCE=boolean
INVALID_BDP_DATA=boolean
STATUS=status variable

Parameters *LOAD_MAP* or *LM*

Default load map file. This file can be positioned. If LOAD_MAP is omitted, the default file is not changed. The initial default file reference is $LOCAL.LOADMAP.$BOI.

LOAD_MAP_OPTIONS or *LOAD_MAP_OPTION* or *LMO*

Set of one or more keywords indicating the information included in the load map. Options are:

NONE

No load map is written.

SEGMENT (S)

Segment map.

BLOCK (B)

Block map.

ENTRY_POINT (EP)

Entry point map.

CROSS_REFERENCE (CR)

Entry point cross-reference.

ALL

Segment map, block map, entry point map, and entry point cross-reference.

If LOAD_MAP_OPTION is omitted, the default load map options are not changed. The initial default option is NONE.

PRESET_VALUE or *PV*

Default value stored in all uninitialized data words. Options are:

ZERO (Z)

All zeros.

FLOATING_POINT_INDEFINITE (FPI)

Floating-point indefinite value.

INFINITY (I)

Floating-point infinite value.

ALTERNATE_ONES (AO)

Alternating 0 and 1 bits; the leftmost (highest order) bit is 1.

If PRESET_VALUE is omitted, the default preset value is not changed. The initial preset value is ZERO.

TERMINATION_ERROR_LEVEL or *TEL*

Default error level that terminates program loading. Options are:

WARNING (W)

Warning, error, or fatal error.

ERROR (E)

Error or fatal error only.

FATAL (F)

Fatal error only.

If TERMINATION_ERROR_LEVEL is omitted, the default termination error level is not changed. The initial termination error level is ERROR.

DEBUG_INPUT or *DI*

Default Debug command file. The commands are read only if the program is executed in debug mode. This file can be positioned. If DEBUG_INPUT is omitted, the default debug input file is not changed. The initial default file is COMMAND.

DEBUG_OUTPUT or *DO*

Default debug output file. Output is written only if the program is executed in debug mode. This file can be positioned. If DEBUG_OUTPUT is omitted, the default debug output file is not changed. The initial default file is $OUTPUT.

ABORT_FILE or *AF*

File containing debug commands to be processed if the program aborts. The commands are used only if the program is not executed in debug mode. This file can be positioned. If ABORT_FILE is omitted, the default abort

file is not changed. The initial default file is $NULL, indicating that no debug commands are processed if the program aborts.

DEBUG_MODE or *DM*

Default debug mode setting. (For information on using Debug, refer to the specific program's source language manual.) Options are:

ON

Program executed under Debug control.

OFF

Program executed without Debug control.

If DEBUG_MODE is omitted, the default debug option is not changed. The initial default value is OFF.

DELETE_LIBRARIES or *DELETE_LIBRARY* or *DL*

List of object libraries to be deleted from the job library list. If you specify ALL, the job library list is cleared. The object libraries are deleted before the object libraries specified on the ADD_LIBRARY parameter are added. The keyword OSF$TASK_SERVICES_LIBRARY specifies the system table.

Specify DELETE_LIBRARY=ALL to reorder your job library list.

If DELETE_LIBRARY is omitted, no object libraries are deleted.

ADD_LIBRARIES or *ADD_LIBRARY* or *AL*

List of object libraries to add to the job library list. The libraries are added to the beginning of the job library list in the order listed. The keyword OSF$TASK_SERVICES_LIBRARY specifies the system table.

Use the ADD_LIBRARY parameter to specify all object libraries in your job library list and their respective order.

To reorder your job library list, specify DELETE_LIBRARY=ALL and then use the ADD_LIBRARY parameter to list the order desired.

If ADD_LIBRARY is omitted, no object libraries are added.

ARITHMETIC_OVERFLOW or *AO*

This parameter specifies whether or not the hardware condition ARITHMETIC_OVERFLOW causes an interrupt. Valid specifications are:

ON

ARITHMETIC_OVERFLOW is enabled. The condition causes an interrupt.

OFF

ARITHMETIC_OVERFLOW is disabled. The condition does not cause an interrupt.

ARITHMETIC_LOSS_OF_SIGNIFICANCE or *ALOS*

This parameter specifies whether or not the hardware condition ARITHMETIC_LOSS_OF_SIGNIFICANCE causes an interrupt. Valid specifications are:

ON

ARITHMETIC_LOSS_OF_SIGNIFICANCE is enabled. The condition causes an interrupt.

OFF

ARITHMETIC_LOSS_OF_SIGNIFICANCE is disabled. The condition does not cause an interrupt.

DIVIDE_FAULT or *DF*

This parameter specifies whether or not the hardware condition DIVIDE_FAULT causes an interrupt. Valid specifications are:

ON

DIVIDE_FAULT is enabled. The condition causes an interrupt.

OFF

DIVIDE_FAULT is disabled. The condition does not cause an interrupt.

EXPONENT_OVERFLOW or *EO*

This parameter specifies whether or not the hardware condition EXPONENT_OVERFLOW causes an interrupt. Valid specifications are:

ON

EXPONENT_OVERFLOW is enabled. The condition causes an interrupt.

OFF

EXPONENT_OVERFLOW is disabled. The condition does not cause an interrupt.

EXPONENT_UNDERFLOW or *EU*

This parameter specifies whether or not the hardware condition EXPONENT_UNDERFLOW causes an interrupt. Valid specifications are:

ON

EXPONENT_UNDERFLOW is enabled. The condition causes an interrupt.

OFF

EXPONENT_UNDERFLOW is disabled. The condition does not cause an interrupt.

FP_INDEFINITE or *FPI* or *FI*

This parameter specifies whether or not the hardware condition FP_INDEFINITE causes an interrupt. Valid specifications are:

ON

FP_INDEFINITE is enabled. The condition causes an interrupt.

OFF

FP_INDEFINITE is disabled. The condition does not cause an interrupt.

FP_LOSS_OF_SIGNIFICANCE or *FPLOS* or *FLOS*

This parameter specifies whether or not the hardware condition FP_LOSS_OF_SIGNIFICANCE causes an interrupt. Valid specifications are:

ON

FP_LOSS_OF_SIGNIFICANCE is enabled. The condition causes an interrupt.

OFF

FP_LOSS_OF_SIGNIFICANCE is disabled. The condition does not cause an interrupt.

INVALID_BDP_DATA or *IBDPD* or *IBD*

This parameter specifies whether or not the hardware condition INVALID_BDP_DATA causes an interrupt. Valid specifications are:

ON

INVALID_BDP_DATA is enabled. The condition causes an interrupt.

OFF

INVALID_BDP_DATA is disabled. The condition does not cause an interrupt.

Remarks

- The program attributes include the object libraries in the job library list and the default execution option values. The job library list is always included in the program library list of each program in the job. You can override the default execution option values if you execute the program with an EXECUTE_TASK command or if you execute a predefined program description module.
- In general, the default program attribute values do not affect system-defined commands. This is because the program descriptions for system-defined commands specify all program attribute values for the command.
- Enter a DISPLAY_PROGRAM_ATTRIBUTES command to display the current job library list and default execution option values.
- By specifying a status variable on the SET_PROGRAM_ATTRIBUTES command, error messages are suppressed. The status variable describes the error.
- When you add libraries to the job library list using the ADD_LIBRARY parameter, the list you specify is processed in reverse order. If an error occurs during this process, the libraries specified in the list after the

point of error are added to the job library list. The library that incurred the error and all libraries preceding it in the list are not added to the job library list.

- For more information, see the NOS/VE Object Code Management manual.

Examples The following command changes the job library list so it contains only libraries LIB1 and LIB2, in that order.

```
/set_program_attributes delete_libraries=all ..
../add_libraries=(lib1,lib2)
```

SET_SENSE_SWITCH Command

Purpose Sets software-managed sense switches of your job or a job you control to either on or off.

Format

SET_SENSE_SWITCH or
SET_SENSE_SWITCHES or
SETSS
JOB_NAME=name
ON=list of range of integer
OFF=list of range of integer
STATUS=status variable

Parameters

JOB_NAME or *JN*

Specifies the name of the job whose sense switches are to be altered. This may be the name you supplied or the system-supplied name. Omission causes the sense switches associated with the requesting job to be altered.

ON

Specifies the switches (from 1 through 8) that are to be turned on. Omission causes no switches to be turned on.

OFF

Specifies the switches (from 1 through 8) that are to be turned off. Omission causes no switches to be turned off.

Remarks For more information, see the NOS/VE System Usage manual.

Examples The following is an example of setting sense switches.

```
/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)
```

SET_WORKING_CATALOG Command

Purpose Establishes a job's default working catalog.

Format

SET_WORKING_CATALOG or
SETWC
CATALOG=file
STATUS=status variable

Parameters

CATALOG or **C**

Specifies the catalog to be used as the working catalog. This can be a specific catalog, the local catalog ($LOCAL), or may specify $USER to indicate the master catalog. This parameter is required.

Remarks

- When a relative file reference (that is, relative path) is given, the names of the working catalog are prefixed to determine a complete file reference.
- Initially, the job's working catalog is $LOCAL.
- For more information, see the NOS/VE System Usage manual.

Examples The following example changes the working catalog to the master catalog.

```
/set_working_catalog $user
/attach_file data_file_1 "$USER path supplied by system."
```

The following example changes the working catalog to subcatalog DATA_CATALOG_1 in the master catalog.

```
/set_working_catalog $user.data_catalog_1
/attach_file data_file_2    "System supplies catalog."
```

The following example displays your working catalog:

```
/display_value $catalog
```

$SEVERITY
Function

Purpose Returns the severity level of the condition you specify as a string.

Format

$SEVERITY
(integer)

Parameters

integer

Specifies the condition code of the status record for which you want the severity level returned. This parameter is required.

Remarks

- The severity level is returned as a string value in uppercase characters. The following are possible values returned:

  INFORMATIVE
  WARNING
  ERROR
  FATAL
  CATASTROPHIC

- For further information about functions, see the NOS/VE System Usage manual.

Examples The following example displays the severity level of a specified condition code:

```
/display_value $severity(pf_status.condition)
ERROR
```

SKIP_TAPE_MARK
Command

Purpose Positions a tape file in a backward or forward direction.

Format

SKIP_TAPE_MARK or
SKIP_TAPE_MARKS or
SKITM
FILE=file
DIRECTION=keyword
COUNT=integer
STATUS=status variable

Parameters

FILE or F

Specifies the local file to be positioned. If the file is not associated with a magnetic tape device, the command is ignored. This parameter is required.

DIRECTION or *D*

Specifies the direction of the file position. The values are:

FORWARD

Positions the file in the forward direction.

BACKWARD

Positions the file in the backward direction.

Omission causes FORWARD to be used.

COUNT or *C*

Specifies the number of tape marks to skip. The file is positioned a specified number of tape marks from the current position until the COUNT is exhausted or a boundary condition is encountered. The boundary condition for a forward skip is the end-of-volume of the last volume in the list of volumes associated with the file. For a backward skip, loadpoint terminates the operation. Omission causes 1 to be used.

Remarks

- This command is not allowed on a labelled tape file.
- For more information, see the NOS/VE System Usage manual.

Examples

The following commands illustrate skipping tape marks.

```
/skip_tape_marks file=master count=3
/skip_tape_marks my_file forward 5
/skip_tape_mark file=new_master direction=backward
```

SORT Command

Purpose Sorts the records from one or more files and writes the records in sorted order to a single output file.

Format

SORT
FROM=list of file
TO=file
KEY=list of any
DIRECTIVES=list of file
LIST=file
LIST_OPTIONS=list of keyword
ERROR=file
ERROR_LEVEL=keyword
RESERVED_POSITION_9=boolean
ESTIMATED_NUMBER_RECORDS=range of integer
EXCEPTION_RECORDS_FILE=file
C170_COMPATIBLE=boolean
OMIT_DUPLICATES=boolean
OWNCODE_FIXED_LENGTH=integer
OWNCODE_MAXIMUM_RECORD_LENGTH=integer
OWNCODE_PROCEDURE_1=name
OWNCODE_PROCEDURE_2=name
OWNCODE_PROCEDURE_3=name
OWNCODE_PROCEDURE_4=name
OWNCODE_PROCEDURE_5=name
RETAIN_ORIGINAL_ORDER=boolean
COLLATING_SEQUENCE_NAME=name
COLLATING_SEQUENCE_STEP=list of any
COLLATING_SEQUENCE_REMAINDER=boolean
COLLATING_SEQUENCE_ALTER=boolean
STATUS=status variable
SUM=list of any
ZERO_LENGTH_RECORDS=keyword
VERIFY_MERGE_INPUT_ORDER=boolean
LOAD_COLLATING_TABLE=list of name
RESULT_ARRAY=integer array

Parameters *FROM* or *F*

List of from 1 through 100 input files.

If you omit the FROM parameter but specify the OWNCODE_PROCEDURE_1 parameter, the sort expects input records from the owncode 1 procedure; otherwise, it attempts to read input records from file OLD in the $LOCAL catalog.

TO or *T*

Output file.

If you omit the TO parameter and specify an owncode 3 procedure that does not return output to the sort, the sort does not require an output file. However, if you omit the TO parameter and specify an owncode 3 procedure that does return output to the sort, the sort writes the output to the default file NEW.

If you omit the TO parameter and the OWNCODE_PROCEDURE_3 parameter, the sort writes all output records to the default file NEW. If file NEW does not exist, the sort creates and uses file $LOCAL.NEW.

KEY or *K*

List of 1 through 106 key field definitions. A key field definition is a value set containing up to four values as follows:

(first..last,type,order) or

(first,length,type,order)

first

Position of the first byte of the key field within the record. (The leftmost byte in a record is numbered 1.)

last

Position of the last byte of the key field within the record.

length

Optional number of bytes in the key field. The default length is 1 byte.

type

Optional name of a numeric data format or collating sequence. The default key field type is ASCII.

The valid numeric data format names are:

BINARY
BINARY_BITS
INTEGER
INTEGER_BITS
NUMERIC_FS
NUMERIC_LO
NUMERIC_LS
NUMERIC_NS
NUMERIC_TO
NUMERIC_TS
PACKED
PACKED_NS
REAL

The predefined collating sequence names are ASCII, ASCII6, COBOL6, DISPLAY, EBCDIC, EBCDIC6.

order

Optional sort order indicator: ascending order (A) or descending order (D). The default is ascending order.

If you omit the KEY parameter, the sort uses one key; the key field extends from the beginning of the record to the smallest minimum record length (MINRL) defined for an input file; the key field has key type ASCII and is sorted in ascending order.

NOTE

If you intend to omit the KEY parameter, you should change the minimum record length attribute value for all input files. If you omit the KEY parameter and have not specified a minimum record length for all files, the sort attempts to use the default minimum record length as the key length. The default minimum record length is 0 (zero); the sort cannot define a key of length 0 (zero) so it returns a fatal error.

DIRECTIVES or *DIRECTIVES_FILE* or *DIR* or *DF*

List of from 1 through 100 directive files.

If you omit the DIRECTIVES parameter, no parameters are read from a directive file; the sort is completely specified on the command.

LIST or *L*

Listing file. If you omit the LIST parameter, the sort listing is written on file $LIST.

LIST_OPTIONS or *LO*

List of one or more list options specifying the additional information to be written to the listing file.

If you omit the LIST_OPTIONS parameter, the sort writes only the minimum information to the listing file (page number, error messages, directives, exception records file summary, and the number of records sorted).

The valid keywords are:

OFF or NONE

No additional information is written to the listing file.

S

Source directives read

DE

Detailed exception information (specify only when EXCEPTION_RECORDS_FILE parameter is specified)

RS

Record statistics

ERROR or *E*

Error message file. If you omit the ERROR parameter, the sort writes any error messages on file $ERRORS.

ERROR_LEVEL or *EL*

Minimum error severity to be reported. If you omit the ERROR_LEVEL parameter, the sort reports all warning, fatal, and catastrophic errors.

The valid keywords are:

INFORMATIONAL (I)

Report all errors

TRIVIAL (T)

Report all errors

WARNING (W)

Report warning, fatal and catastrophic errors only

FATAL (F)

Report fatal and catastrophic errors only

CATASTROPHIC (C)

Report catastrophic errors only

NONE

Report no errors

RESERVED_POSITION_9

Reserved.

ESTIMATED_NUMBER_RECORDS or *ENR*

Integer range from 1 through 16777215. You can specify the ESTIMATED_NUMBER_RECORDS parameter, but the sort does not use the parameter value.

EXCEPTION_RECORDS_FILE or *ERF*

File to which invalid records are written (invalid records are not written to the output file). If you omit this parameter, the sort does not perform exception processing; invalid records are written to the output file.

C170_COMPATIBLE or *CC*

Specifies whether lowercase letters in owncode procedure names are to be converted to uppercase letters. required for loading of the owncode procedure. If you omit this parameter, the default is OFF and the owncode procedure names are not converted. Use YES, TRUE, or ON to specify a logical true.

OMIT_DUPLICATES or *OD*

Specifies whether sort outputs only one record in each set of records with equivalent key values. Duplicates are not omitted; equivalent key values are processed as specified by the OWNCODE_PROCEDURE_5, RETAIN_ORIGINAL_ORDER, or SUM parameter.

TRUE, YES, or NO

Duplicates are omitted.

FALSE, NO, or OFF

Duplicates are not omitted.

OWNCODE_FIXED_LENGTH or *OWNFL* or *OFL*

Fixed record length (integer from 1 through 4096). If you omit the OWNCODE_FIXED_LENGTH parameter, the sort uses the OWNCODE_MAXIMUM_RECORD_LENGTH parameter value as the record length. Specification of this parameter or the OWNCODE_MAXIMUM_RECORD_LENGTH parameter is required if the TO and FROM parameters are omitted.

OWNCODE_MAXIMUM_RECORD_LENGTH or *OWNMRL* or *OMRL*

Maximum record length (integer from 1 through 4096). If you omit the OWNCODE_MAXIMUM_RECORD_LENGTH parameter, the sort uses the record length attribute of the first input file read or output file written as the maximum record length. Specification of this parameter or the OWNCODE_FIXED_RECORD_LENGTH parameter is required if the TO and FROM parameters are omitted.

OWNCODE_PROCEDURE_1 or *OP1* or *OWN1*

Object library entry point name for the procedure that preprocesses each input record. If you omit the OWNCODE_PROCEDURE_1 parameter, no user-defined input record preprocessing is performed.

OWNCODE_PROCEDURE_2 or *OP2* or *OWN2*

Object library entry point name for the procedure that can insert a record after each input file is read. If you omit the OWNCODE_PROCEDURE_2 parameter, no user-defined input file postprocessing is performed.

OWNCODE_PROCEDURE_3 or *OP3* or *OWN3*

Object library entry point name for the procedure that postprocesses each output record. If you omit the OWNCODE_PROCEDURE_3 parameter, no user-defined output record postprocessing is performed.

OWNCODE_PROCEDURE_4 or *OP4* or *OWN4*

Object library entry point name for the procedure that can insert a record at the end of the output file. If you omit the OWNCODE_PROCEDURE_4 parameter, no user-defined output file postprocessing is performed.

OWNCODE_PROCEDURE_5 or *OP5* or *OWN5*

Object library entry point name for the procedure called if two input records have equal key values. This parameter cannot be specified with the SUM parameter. If you omit the OWNCODE_PROCEDURE_5 parameter, no user-defined processing of equal records is performed.

RETAIN_ORIGINAL_ORDER or *ROO* or *RETAIN* or *RET*

Reserved.

COLLATING_SEQUENCE_NAME or *CSN* or *SEQN*

Name of the collating sequence defined by the subsequent collating sequence parameters. If you omit the COLLATING_SEQUENCE_NAME parameter, no user collating sequence is defined.

COLLATING_SEQUENCE_STEP or *CSS* or *SEQS*

List of from 1 through 256 value step definitions. If you omit the COLLATING_SEQUENCE_STEP parameter, no specific collating sequence value steps are defined.

A value step definition is a value set that defines one or more value steps in the collating sequence as follows (char is an ASCII character):

('char')

One 1-character value step.

('char','char',...)

One multicharacter value step.

('char'..'char')

Multiple 1-character value steps.

('char'..'char', 'char'..'char',...)

Multiple multicharacter value steps. (All ranges must be the same size.)

COLLATING_SEQUENCE_REMAINDER or *CSR* or *SEQR*

Indicates whether a special value step is defined containing all characters not specified by other value step definitions. If you omit the COLLATING_SEQUENCE_REMAINDER parameter, the remaining characters keep their same collating positions as in the default ASCII collating sequence.

COLLATING_SEQUENCE_ALTER or *CSA* or *SEQA*

Indicates whether all characters in a value step are altered to match the first character in the value step. If you omit the COLLATING_SEQUENCE_ALTER parameter, no character alteration is performed.

SUM or *S*

List of from 1 through 100 sum field definitions. This parameter cannot be specified with the OWN5 or RETAIN parameters. If you omit the SUM parameter, no summing is performed.

A sum field definition is a value set containing up to four values as follows:

(first..last,type,repeat_count) or

(first,length,type,repeat_count)

first

Position of the first byte of the sum field within the record. (The leftmost byte in a record is numbered 1.)

last

Position of the last byte of the sum field within the record.

length

Optional number of bytes in the sum field. The default length is 1 byte.

type

Name of a numeric data format. The default format is INTEGER. The valid numeric data format names are:

BINARY
BINARY_BITS
INTEGER
INTEGER_BITS
NUMERIC_FS
NUMERIC_LO
NUMERIC_LS
NUMERIC_NS
NUMERIC_TO
NUMERIC_TS
PACKED
PACKED_NS

repeat_count

Optional integer specifying the number of consecutive sum fields defined by the value set. The default number is 1.

ZERO_LENGTH_RECORDS or *ZERO_LENGTH_RECORD* or *ZLR*

Specifies the disposition of zero-length input records. This parameter applies only to records read from input files; it does not apply to records supplied by owncode procedures.

The keywords specifying the disposition of all zero-length input records read for the sort are as follows:

DELETE

Each zero-length record is deleted from the sort or merge. It is not written to the exception records file.

PAD

Each zero-length record is processed as a short record.

LAST

Each zero-length record is written at the end of the output.

VERIFY_MERGE_INPUT_ORDER or *VERIFY* or *VMIO* or *VER*

Specifies that merge should check the order of all merge input records. The records must be in sorted order. If this parameter is omitted, the record order is not checked.

LOAD_COLLATING_TABLE or *LCT*

Loads a collation table to be used by one or more keys. The parameter specifies two values enclosed in parentheses. The first value is the keytype name you specify in the key field definition. The second value is the name of the collation table. The table can be one of the NOS/VE predefined collation tables or a user-defined collation table.

RESULT_ARRAY or *RA* or *RESA*

Specifies an SCL variable in which sort statistics are returned. The variable must be a 16-element integer array. The first element of the array must be assigned the number of statistics to be returned (0 through 15).

Remarks

- A user-defined collating sequence can be defined by a sequence of parameters. The first parameter in the sequence must be a COLLATING_SEQUENCE_NAME parameter naming the collating seqence.

  The naming parameter is followed by one or more COLLATING_SEQUENCE_STEP, COLLATING_SEQUENCE_REMAINDER, and COLLATING_SEQUENCE_ALTER parameters defining the collating sequence steps that differ from the default ASCII collating sequence.

  The collating sequence definition ends when a parameter other than COLLATING_SEQUENCE_STEP, COLLATING_SEQUENCE_REMAINDER, or COLLATING_SEQUENCE_ALTER is read.

  More than one collating sequence can be defined for the sort.

- For more information, see the NOS/VE Advanced File Management Usage manual.

Examples

This command sorts the records on file $LOCAL.OLD and writes the sorted records on file $LOCAL.NEW. The records are sorted using the leftmost 10 characters as the key; the keys are sorted in ascending order using the ASCII collating sequence.

```
/sort,key=((1,10))
```

This command sorts the records on permanent files FILE1 and FILE2 and writes the sorted records on permanent file FILE3. The records are sorted using the leftmost 60 bits as the key; the keys are sorted in descending order as integer values.

```
/sort,($user.file1,$user.file2),$user.file3,..
../key=((1..60,integer_bits,d))
```

This command sorts the records on FILE4 and writes the sorted records on FILE5. The records are sorted using two keys: the first key is bytes 9 through 16 of the record sorted in descending order as integer values; the second key is the third character of the record sorted in ascending order using the ASCII collating sequence. The first key takes precedence over the second key.

```
/sort,file4,file5,((9..16,integer,d),(3))
```

SOURCE_CODE_UTILITY Command

Purpose

Begins an SCU command utility session.

Format

SOURCE_CODE_UTILITY or
SCU or
SOUCU
STATUS=status variable

Remarks

- Entering a CREATE_LIBRARY or USE_LIBRARY subcommand initializes the working library for the SCU command utility session. If neither subcommand is issued, file SOURCE_LIBRARY is used for the base and result libraries. If file SOURCE_LIBRARY does not exist, it is created.

- For more information, see the NOS/VE Source Code Management manual.

Examples The following sequence begins an SCU session and initializes the working library from file OLDPL in your working catalog, assumed not to be $LOCAL. The base file, OLDPL, is a source file whose file structure is a library. Entering the QUIT subcommand causes the working library to be written on the next cycle of file OLDPL.

```
/source_code_utility
sc/use_library base=oldpl result=oldpl.$next
sc/quit
```

The next example does not use the USE_LIBRARY subcommand, but rather initializes the working library from file SOURCE_LIBRARY in your working catalog.

```
/source_code_utility
sc/create_deck deck=deck1 ..
sc../modification=version1
sc/quit
```

$SPECIFIED
Function

Purpose Returns a boolean value indicating whether a parameter was specified in the parameter list and passed to the procedure, or whether the default value was passed to the procedure.

Format

$SPECIFIED
(name)

Parameters

name

Specifies the parameter you are interrogating. This parameter is required.

Remarks

- This function is used to reference parameters within procedures.
- If this function returns a value of TRUE, the named parameter was specified on the parameter list and passed to the procedure. If a value of FALSE is returned, the named parameter was not specified, and the default value, if any, was passed to the procedure.
- For further information about functions, see the NOS/VE System Usage manual.

Examples The following example shows the portion of a procedure that determines whether its OUTPUT parameter was specified in its parameter list:

```
IF $specified(output) THEN
    .
    .   "Execute if OUTPUT specified.
    .
ELSE
    .
    .   "Execute if OUTPUT not specified.
    .
IFEND
```

$STATUS Function

Purpose Returns a status value with the status record fields you specify.

Format

$STATUS
(boolean
string1
any
string2
string3
string4
string5
string6
string7
string8
string9
string10
string11)

Parameters

boolean

Specifies the boolean value of the NORMAL field of a status record. If the normal parameter is TRUE, the remaining parameters are ignored. If the value of this parameter is FALSE, you must supply either the condition parameter or the condition and identifier parameters. This parameter is required.

string1

Specifies the IDENTIFIER field of the status record. It is a 2-character string.

any

Specifies the CONDITION field of the status record. It is either an integer or the name of a status condition.

The condition parameter is assumed to represent the complete condition code for $STATUS when its integer value is greater than 0FFFFFF(hexadecimal). When the CONDITION parameter is less than or equal to this value, you must also specify the product identifier parameter.

string2

Specifies the TEXT field of the status record. It is a string of up to 256 characters containing a maximum of 10 separate message parameters. The default is a null string.

A unit separator character is prefixed to each message parameter prior to its being inserted in the TEXT field. This character is used as the status parameter delimiter.

string3

Reserved.

string4

Reserved.

string5

Reserved.

string6

Reserved.

string7

Reserved.

string8

Reserved.

string9

Reserved.

string10

Reserved.

string11

Reserved.

Remarks

- Some condition code numbers are reserved for Control Data internal users; other numbers are reserved for external customer use. For information on condition code numbers, refer to the NOS/VE Object Code Management manual.
- For further information about functions, see the NOS/VE System Usage manual.

Examples

- The following example displays a constructed status value with two message parameters:

```
/display_value $status(false,'CL', ..
../cle$wrong_kind_of_value,..
../ 'integer', 'string')
 --ERROR-- Expecting integer value, found string.
```

- The following example illustrates what is displayed if the system cannot find the status message associated with the status condition:

```
/display_value $status(false, 'MY',0,'param1',..
../'param2')
--ERROR--  CC=MY 0 TEXT=?param1?param2
```

- The $STATUS function is especially useful when used to report the terminating condition of an SCL procedure. The EXIT_PROC statement can be used to exit a procedure with a specified status, as shown in the following example:

```
EXIT_PROC WITH $status(false, ..
'CL',cle$wrong_kind_of_value,..
'INTEGER', 'STRING')
```

$STRING Function

Purpose Converts a file reference, name, or variable reference to an uppercase string.

Format

$STRING
(any)

Parameters

any

Identifies the file reference, name, or variable reference to be converted. This parameter is required.

Remarks

- If you are converting a file to a string, you can specify how the file is to be positioned prior to use.
- This function is useful when it is necessary to compare two names. Because names cannot be directly compared with relational operators, they must first be converted to strings and the resulting strings compared.
- For further information about functions, see the NOS/VE System Usage manual.

Examples The following example uses the $VALUE function to compare two file names that were passed to an SCL procedure:

```
IF $string($value(from)) = $string($value(to)) THEN
    .
    .   "Issue appropriate status and exit procedure."
    .
IFEND
```

The $VALUE function returns the actual value that was specified for a procedure parameter. The example assumes that a procedure was called with file parameters FROM and TO. The $VALUE(FROM) function supplies the actual value specified in the FROM parameter. The $STRING function then converts that value (a name) to a string that can be compared with the string returned by $STRING($VALUE(TO)). This string contains the name specified in the TO parameter.

If the names passed to the procedure are identical, the procedure sets an appropriate status and terminates.

$STRLEN Function

Purpose Returns an integer that represents the current length of a string.

Format

$STRLEN
(string)

Parameters

string

Specifies the string whose length you are interrogating. This parameter is required.

Remarks

- The length returned is the current length, not the maximum defined length. The current length is equal to the character position of the last character in a string.
- For further information about functions, see the NOS/VE System Usage manual.

Examples In the following example, the system is interrogated for the current length of string 'abc//'123':

```
/display_value $strlen('abc'//'123')
6
```

$STRREP Function

Purpose Converts an integer, boolean value, file, name, real value, status, or variable to a string.

Format

$STRREP
(any
***integer*)**

Parameters

any

Specifies the integer, boolean value, file, name, real value, status, string, or variable you wanted converted to a string. This parameter is required.

integer

Specifies the radix associated with an integer value. The radix is not part of the created string value. The default radix is decimal.

Remarks

- If you supply a file reference as the argument for this function, the file path returned depends on the current message mode. If the current message mode if FULL, the complete file path is returned. If the current message mode is BRIEF, the file path relative to the working catalog is returned.

  You can use the SET_MESSAGE_MODE command to change the current message mode.

- For further information about functions, see the NOS/VE System Usage manual.

Examples

- The following example converts the integer 256 to hexadecimal and displays the resulting string:

```
/display_value $strrep(256,16)
0FF
100
```

- The following example converts the decimal integer 512 to octal.

```
/display_value '512 decimal is the same as '..
..///$strrep(512,8)//' octal.'
512 decimal is the same as 1000 octal.
```

- The following example converts the boolean value TRUE to its string representation in uppercase characters:

```
/display_value 'The result is '//$strrep(true)//'.'
The result is TRUE.
```

- The following example converts the hexadecimal number 0FF to decimal:

```
/disv $strrep(0ff)(16))
255
```

For an alternative way of converting integers, see the description of the DISPLAY_VALUE command.

- In the following example, the $STRREP function converts into a string any type of value you specify for SUBJECT:

```
PROC exp, e (
  subject : any = 'GETTING HELP'
  manual  : file = $system.manuals.scl
  status  : var of status = $optional
  )

    explain s=$strrep($value(subject)) ..
    m=$value(manual)

PROCEND exp
```

When called, the procedure executes the following command:

```
explain s='ACCL' m=$system.manuals.scl
```

SUBMIT_JOB Command

Purpose Submits a job to NOS/VE for batch processing.

Format

SUBMIT_JOB or
SUBJ
FILE=file
CPU_TIME_LIMIT=integer or *keyword*
JOB_ABORT_DISPOSITION=keyword
JOB_CLASS=name
JOB_DESTINATION=any
JOB_DESTINATION_USAGE=name or *keyword*
JOB_EXECUTION_RING=integer
JOB_QUALIFIER=list of name or *keyword*
JOB_RECOVERY_DISPOSITION=keyword
LOGIN_FAMILY=name
MAGNETIC_TAPE_LIMIT=integer or *keyword*
MAXIMUM_WORKING_SET=integer or *keyword*
OPERATOR_FAMILY=name
OPERATOR_USER=name
OUTPUT_DISPOSITION=file or *keyword*
REMOTE_HOST_DIRECTIVE=string

SRU_LIMIT=integer or *keyword*
STATION=name or *keyword*
USER_INFORMATION=string
USER_JOB_NAME=name
SYSTEM_JOB_NAME=string variable
STATUS=status variable

Parameters

FILE or **F**

Specifies the file to be submitted for processing as a batch job. This parameter is required.

CPU_TIME_LIMIT or *CTL*

Specifies the maximum cpu time in seconds that will be allocated to the job. If the value specified is greater than the maximum cpu time limit allowed for the user by the site, the job will be terminated immediately. During job execution, if the job's accumulated job and monitor cpu time exceeds the value specified here, a job abort limit condition will occur and the job will be terminated.

If this parameter is specified, the job classes that the job can be a member of may be restricted. The job may only be a member in a job class that supports a cpu time limit greater than or equal to this parameter's value.

Specify one of the following on this parameter:

integer

Maximum cpu seconds allocated for the job.

SYSTEM_DEFAULT

The system default value for this attribute is to be used.

UNLIMITED

The maximum value allowed by the system is to be used.

UNSPECIFIED

Assigns the value that is used when the parameter is omitted.

If this parameter is omitted and an explicit job class name is specified on the JOB_CLASS parameter of this command, the cpu time limit is determined from the job class' cpu time limit or the user's validation cpu time

limit, whichever is less. If the JOB_CLASS is AUTOMATIC, the cpu time limit for the job is determined by the system default value or the user's validation cpu time limit, whichever is less.

This parameter takes precedence over the CPU_TIME_LIMIT parameter on the LOGIN command.

JOB_ABORT_DISPOSITION or *JAD*

Specifies what should be done with the job if it aborts because of system failure. This parameter takes precedence over the JOB_ABORT_DISPOSITION parameter on the LOGIN command. If neither the LOGIN command nor this command specifies this parameter, then the system default is used. The keywords are:

RESTART

Job is automatically resubmitted so that it starts over from the beginning.

TERMINATE

Job is discarded.

JOB_CLASS or *JC*

Specifies the name of the job class to be used for the job. This parameter takes precedence over a JOB_CLASS parameter on the LOGIN command. If neither this command nor the LOGIN command specifies a job class, the default job class of the login user is used. If a default job class does not exist for the login user, the system default is used.

The name AUTOMATIC (if the user is validated to use the job class AUTOMATIC) will automatically assign the job to a job class based on the job's attributes.

JOB_DESTINATION or *JD*

Specifies the location name of the system where the job is to execute. 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.

This parameter is required if the JOB_DESTINATION_USAGE parameter is specifed. If this parameter is specified and the JOB_DESTINATION_USAGE parameter is not specified, QTF is assumed for the JOB_DESTINATION_USAGE parameter.

JOB_DESTINATION_USAGE or *JDU*

Specifies the queue file transfer application to be used to forward the job to a remote system for execution. If this parameter is specified, the job is placed in the job store-and-forward queue and assigned to the named queue file transfer application for forwarding. This occurs even if the job destination has the same name as a local family.

If this parameter is specified, the JOB_DESTINATION parameter must also be specified. If this parameter is omitted and the JOB_DESTINATION parameter is specified, the job is assigned to the QTF application for forwarding. This occurs even if the job destination has the same name as a local family.

If both this parameter and the JOB_DESTINATION parameter are omitted and the LOGIN_FAMILY parameter specifies a local family, the job is prevalidated and placed in the local job queue. If both this parameter and the JOB_DESTINATION parameter are omitted and the LOGIN_FAMILY parameter does not specify a local family, the job is assigned to the QTF application for forwarding.

The keywords are:

NTF

The NTF queue file transfer application is used to forward the job. See your site personnel for more information on the NTF application.

QTF

The QTF queue file transfer application is used to forward the job. See the RHF Usage manual for more information on the QTF application.

JOB_EXECUTION_RING or *JER*

Specifies the job's execution ring. Allowable values are from 4 through 13, but must be greater than or equal to the user's minimum ring validation. This parameter takes precedence over the JOB_EXECUTION_RING parameter on the LOGIN command. If neither this command nor the LOGIN command specifies a job execution ring, the user's nominal ring is used.

JOB_QUALIFIER or *JOB_QUALIFIERS* or *JQ*

Specifies from one to five site-defined names used to possibly limit a job to a specific job class or set of classes or mainframes.

Specify one of the following:

list of names

Up to five site defined job qualifiers.

NONE

No job qualifiers are to be used.

SYSTEM_DEFAULT

The system default value for this attribute is to be used.

If this parameter is omitted, the system default value is used.

JOB_RECOVERY_DISPOSITION or *JRD*

Specifies what should be done with the job by the active job recovery process if there is a system interrupt while the job is executing. This parameter takes precedence over the JOB_RECOVERY_DISPOSITION parameter on the LOGIN command. If neither the LOGIN command nor this command specifies this parameter, the system default is used. 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 continues normal execution. If the attempt fails, the value specified on the JOB_ABORT_DISPOSITION parameter is used.

RESTART

Job is automatically resubmitted so that it starts over from the beginning.

TERMINATE

Job is discarded.

LOGIN_FAMILY or *FAMILY_NAME* or *FN* or *LF*

Specifies the family name under which the job is to be run. This parameter takes precedence over the LOGIN_FAMILY parameter on the LOGIN command. If neither this command nor the LOGIN command specifies this parameter, the family of the submitting job is used if the job is to be run locally and the remote system default family is used if the job is to be run at a remote NOS/VE system.

MAGNETIC_TAPE_LIMIT or *MTL*

Specifies the maximum number of magnetic tape drives required simultaneously by the job.

If this parameter is specified, the job classes that the job can be a member of may be restricted. The job may only be a member in a job class that supports a magnetic tape limit greater than or equal to this parameter's value.

Specify one of the following on this parameter:

integer

Maximum number of tape units required by the job.

SYSTEM_DEFAULT

The system default value for this attribute is to be used.

UNLIMITED

The maximum value allowed by the system is to be used.

UNSPECIFIED

Assigns the value that is used when the parameter is omitted.

If this parameter is omitted, the system default is used.

This parameter takes precedence over the MAGNETIC_TAPE_LIMIT parameter on the LOGIN command.

MAXIMUM_WORKING_SET or *MAXWS*

Specifies the maximum working set in pages that the job requires. If this parameter is specified, the job classes that the job can be a member of may be restricted. The

job may only be a member in a job class that supports a maximum working set size greater than or equal to this parameter's value.

Specify one of the following on this parameter:

integer

Maximum working set in pages needed for the job.

SYSTEM_DEFAULT

The system default value for this attribute is to be used.

UNLIMITED

The maximum value allowed by the system is to be used.

UNSPECIFIED

Assigns the value that is used when the parameter is omitted.

If this parameter is omitted, this values is determined by the job class specified on the JOB_CLASS parameter of this command. If the JOB_CLASS parameter is also not specified, the system default value is used.

This parameter takes precedence over the MAXIMUM_WORKING_SET parameter on the LOGIN command.

OPERATOR_FAMILY or *OF*

Specifies the default private station or remote system operator family name attribute for output files generated by this job. If the OUTPUT_DESTINATION_USAGE value for an output file is PRIVATE or NTF, 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 user attribute of output files with OUTPUT_DESTINATION_USAGE values of PRIVATE or NTF.

OPERATOR_USER or *OU*

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.

OUTPUT_DISPOSITION or *STANDARD_OUTPUT* or *SO* or *ODI*

Specifies how to dispose 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. If this parameter is omitted, PRINTER is used.

file_name

Specification of a file name indicates that the standard output is to be copied to the specified permanent file at job end. You may not specify a remote family name with this file name.

DISCARD_ALL_OUTPUT (DAO)

All output files generated by the job are discarded. This option has no effect unless the job destination is a NOS/VE or NTF system.

DISCARD_STANDARD_OUTPUT (DSO)

Standard output is to be 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 the local system, this option causes 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)

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 for the output file name or the user file name for output generated by the PRINT_FILE command. If the $WAIT_QUEUE subcatalog does not exist at the time the output files are returned, it will be created for the user.

REMOTE_HOST_DIRECTIVE or *RHD*

Specifies a default text string which may be used to control processing of jobs submitted to remote systems. 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 JOB_DESTINATION_USAGE parameter specifies the appropriate value. For more information on submitting jobs to remote systems, see the NOS/VE System Usage manual.

If omitted, the REMOTE_HOST_DIRECTIVE job attribute is used.

SRU_LIMIT or *SL*

Specifies the maximum system resource units (srus) that will be allocated to the job. If the value specified is greater than the maximum sru limit allowed for the user by the site, the job will be terminated immediately. During job execution, if the job's accumulated srus exceed the value specified here, a job abort limit condition will occur and the job will be terminated.

If this parameter is specified, the job classes that the job can be a member of may be restricted. The job may only be a member in a job class that supports an sru limit greater than or equal to this parameter's value.

Specify one of the following on this parameter:

integer

Maximum srus allocated for the job.

SYSTEM_DEFAULT

The system default value for this attribute is to be used.

UNLIMITED

The maximum value allowed by the system is to be used.

UNSPECIFIED

Assigns the value that is used when the parameter is omitted.

If this parameter is omitted and an explicit job class name is specified on the JOB_CLASS parameter of this command, the sru limit is determined from the job class' sru limit or the user's validation sru limit, whichever is less. If the JOB_CLASS is AUTOMATIC, the sru limit for the job is determined by the system default value or the user's validation sru limit, whichever is less.

This parameter takes precedence over the SRU_LIMIT parameter on the LOGIN command.

STATION or *S*

If the JOB_DESTINATION_USAGE parameter does not specify NTF, this parameter specifies the default I/O station name for output files generated by the job.

If the JOB_DESTINATION_USAGE parameter does specify NTF, this parameter specifies the name of the control facility that controls the transfer of the job to the NTF remote system. See your site personnel for more information on the NTF application.

If the JOB_DESTINATION_USAGE attribute specifies PRIVATE, this parameter must specify the control facility name.

The keyword AUTOMATIC, indicates that the system default is to be used.

USER_INFORMATION or *UI*

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.

If omitted, the user information string associated with the submitting job is assumed.

USER_JOB_NAME or *JOB_NAME* or *JN* or *UJN*

Specifies the name by which the submitted job is to be known. This name is used in place of a user job name specified as a parameter on the LOGIN command. If this parameter is omitted and the LOGIN command does not specify a USER_JOB_NAME, the LOGIN_USER value from the LOGIN command is used.

SYSTEM_JOB_NAME or *SJN*

Specifies a variable to which the system-supplied name of the job is returned. The system-supplied name is returned in a string variable.

Remarks

- For jobs whose destination is NOS/VE, The batch job is the collection of SCL statements contained in a file. A LOGIN command must be included as the first command in the file.
- The progress of the jobs that execute on the local system can be determined using the DISPLAY_JOB_STATUS and DISPLAY_HISTORY_LOG commands.
- If both the JOB_DESTINATION and LOGIN_FAMILY are specified on this command, the job first goes to the job destination. Once there, the LOGIN_FAMILY value is examined to see if it is a family that is local or remote to the job destination system. If the family is remote to the job destination system, QTF is used to transfer the job to the remote system.
- The file referenced on the FILE parameter of this command must have a FILE_CONTENTS file attribute of UNKNOWN or LIST.
- For more information, see the NOS/VE System Usage manual.

Examples In the following example, a NOS/VE job is created on file SUBMIT_FILE using the COLLECT_TEXT command.

```
/colt submit_file

ct? login login_user=sdh password=pass456 ..
  login_family=nve
ct? display_command_list all
ct? logout
ct? **
```

The job is complete with LOGIN and LOGOUT commands (the LOGOUT command is optional). The job requests a full display of the command list. The job is submitted with the following command and is given the name MY_JOB.

```
/submit_job f=submit_file ujn=my_job
```

You can monitor the progress of the job using the DISPLAY_JOB_STATUS command. For example:

```
/disjs jn=my_job do=all
Control_Family                : nve
Control_User                  : sclqr
CPU_Time_Used                 : Job Mode- 16.137
                                Monitor Mode- 0.365
Display_Message               : forend
Job_Class                     : batch
Job_Destination_Usage         : ve
Job_Mode                      : batch
Job_State                     : initiated
Login_Family                  : nve
Login_User                    : sclqr
Operator_Action_Posted        : no
Page_Faults                   : Assigned- 105
                                From Disk- 67
                                Reclaimed- 23
System_Job_Name               : $0990_0102_aad_2011
User_Job_Name                 : my_job
```

The preceding display indicates that the job is executing (the Display_Message field shows the command currently executing). A subsequent DISPLAY_JOB_STATUS command could not locate the job, which indicates that the job has completed.

```
/disjs jn=my_job do=all
Name Not Found          :  my_job
```

$SUBSTR Function

Purpose Returns a specified portion of a string.

Format

$SUBSTR
(string1
integer1
integer2
string2)

Parameters

string1

Specifies the string from which you want the substring extracted. This parameter is required.

integer1

Specifies the position in the string where you want the substring to begin. This parameter is required.

integer2

Specifies the length (in characters) of the substring you want returned. The default length is 1.

string2

Specifies the character used to pad the string when the full string is shorter than the requested length of the substring. The substring is padded on the right with the fill character. The default is the ASCII space character.

Remarks

- To use $SUBSTR in its simplest form, provide a string and a substring starting position. $SUBSTR returns th single character that occurs at the specified position.
- For further information about functions, see the NOS/VE System Usage manual.

Examples

- The following example accepts the default length of 1 character and the substring starting position. $SUBSTR returns the single character that occurs at the specified position.

```
/display_value $substr('abcdefghijklm',10)
j
```

- The next example specifies a substring length of 3 characters. It returns a 3-character string starting with the 10th character of the string 'abcdefghijklm'.

```
/display_value $substr('abcdefghijklm',10,3)
jkl
```

- The next example specifies the character that fills the substring when the length you request is longer than the string. It returns an 8-character substring starting with the 10th character of the string 'abcdefghijklm' and specifies the fill character to be "-".

```
/display_value $substr('abcdefghijklm',10,8,'-')
jklm----
```

- In the next example, the $DATE function is provided as the string subject in the $SUBSTR function. The current Julian date is returned.

```
/display_value $date(ordinal)
1987087
/display_value $substr($date(ordinal),5,3)
087
```

TABLEND UTILITY Subcommand

Purpose Ends the collection of definitions for a utility command table. This command has no parameters.

Format TABLEND

Parameters None.

Remarks

- This command separates the entries in the command table from the executable statements in the established utility.

  This command is required only if the utility's command table is defined within the UTILITY/UTILITYEND block.

- For more information, see the NOS/VE System Usage manual.

TASK Command

Purpose Delimits a sequence of one or more SCL statements to be executed as a synchronous or asynchronous task.

Format

TASK
TASK_NAME=name
RING=integer
DEBUG_MODE=boolean
SUBSTITUTION_MARK=string or *keyword*
STATUS=status variable

Parameters

TASK_NAME or *TN*

Specifies that the delimited command sequence is to be executed as an asynchronous task and provides a name that can be used to refer to the task.

RING or *R*

Specifies the ring at which the new task is to begin execution. This parameter can be used to switch to any ring between and including the minimum ring for which you are validated and 13.

If RING is omitted, the current ring is used.

DEBUG_MODE or *DM*

Indicates whether the task is to be run in Debug mode. Options are:

ON

Task executed under Debug control.

OFF

Task executed without Debug control.

If the DEBUG_MODE is omitted, OFF is used.

SUBSTITUTION_MARK or *SM*

Specifies a one-character string used within a statement to delimit the text to be substituted. This character cannot be any character that is valid in an SCL name.

Corresponding pairs of substitution marks must appear on the same line.

If a second substitution mark is not found on the same line, the end-of-line is treated as the second mark.

If two consecutive substitution marks appear, they are replaced by a single substitution mark in the text.

Substitution marks cannot be used to construct the TASKEND statement.

Remarks

- An asychronous task inherits the current system environment objects of the job. Any change to these objects by the asychronous task are effective only for the asychronous task. The system environment objects are COMMAND_LIST, FILE_CONNECTIONS, INTERACTION_STYLE, MESSAGE_LEVEL, NATURAL_LANGUAGE, PROGRAM_ATTRIBUTES, and WORKING_CATALOG.
- The commands (statement list) to be processed as a separate task are specified between the TASK and TASKEND statements.
- For more information, see the NOS/VE System Usage manual.

$TASK_NAME Function

Purpose Returns the name of the requesting task as a string.

Format **$TASK_NAME**

Parameters None.

Remarks

- The task name is the name supplied on either the TASK/TASKEND statements or the EXECUTE_TASK command that initiated the requesting task. If the requesting task was not given a name in this way, $TASK_NAME returns a null string.
- For further information about functions, see the NOS/VE System Usage manual.

Examples The following example creates a task and displays its own task name:

```
/task alpha
task/display_value $task_name
task/taskend
/
ALPHA
```

$TASK_STATUS Function

Purpose Returns status information for a task that was executed asynchronously and was specified by its task name.

Format

$TASK_STATUS
(name,
keyword)

Parameters

name

Specifies the task (supplied on either the TASK/TASKEND statements or the EXECUTE_TASK command that initiated the requesting task). This parameter is required.

keyword

Specifies the type of status information you wanted returned. Use one of the following values:

COMPLETED (C)

Asks whether the task is known to the system and has not yet terminated. The value returned is either TRUE (task is known to the system and has not yet terminated) or FALSE.

STATUS (S)

Returns the completion status of the task. If the task has terminated, its final status is returned. If the task has not yet terminated, or if it is not known to the system, normal status is returned.

Remarks For further information about functions, see the NOS/VE System Usage manual.

Examples The following example shows that TASK_2 is known to the system and has not yet terminated:

```
/$task_status(task_2,complete)
FALSE
```

$TERMINAL_MODEL Function

Purpose Returns the terminal model number.

Format

$TERMINAL_MODEL
(file)

Parameters *file*

Specifies the name of a terminal file associated with the terminal.

Remarks

- For this function to return the model of your terminal, the model must first be set with the CHANGE_TERMINAL_ATTRIBUTES command.
- The model number is returned as a string. If no terminal model is defined, a null string is returned.
- For further information about functions, see the NOS/VE System Usage manual.

Examples The following example queries the system for the terminal model name:

```
/display_value $terminal_model
DEC_VT220
```

TERMINATE_COMMAND Command

Purpose Terminates all program activity you invoked that was interrupted because of a pause break.

Format

TERMINATE_COMMAND or
TERC
STATUS=status variable

Remarks

- This command is valid only while activity is suspended after a pause break. All activity in the job is terminated and further commands can be entered.
- For more information, see the NOS/VE System Usage manual.

Examples

In this example, the SET_PASSWORD command is suspended, some other commands entered, and then the command is terminated with the following sequence.

```
Enter new password
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXX      Pause break entered.
*Suspended -  1*
p/ "Perform some other activity"
p/terminate_command
Command terminated.
/
```

TERMINATE_JOB Command

Purpose

Terminates one or more jobs.

Format

TERMINATE_JOB or
TERMINATE_JOBS or
TERJ
NAME=list of name
JOB_STATE=keyword
OUTPUT_DISPOSITION=keyword
STATUS=status variable

Parameters

NAME or **NAMES** or **JOB_NAME** or **JOB_NAMES** or **JN** or **N**

Specifies the names of the jobs to be terminated. This may be the names you supply or the system-supplied names. This parameter is required.

JOB_STATE or *STATE* or *S* or *JS*

Restricts the termination of the jobs to those in the specified state. The keywords are:

ALL

Uses all of the following keyword values.

DEFERRED (D)

Jobs not yet eligible to be initiated.

QUEUED (Q)

Jobs waiting to be initiated.

INITIATED (I)

Jobs that have been initiated.

TERMINATED (T)

Jobs that are terminating.

OUTPUT_DISPOSITION or *ODI*

Specifies how the terminated job's standard output is to be disposed of. The following list describes the results of each of the allowable values. If this parameter is omitted, the output disposition attribute of the terminating job is unchanged.

DISCARD_STANDARD_OUTPUT (DSO)

The job's standard output file is to be discarded.

PRINTER (P)

The job's standard output file is to be printed.

WAIT_QUEUE (WQ)

The job's standard output file is to be copied to the user's $WAIT_QUEUE subcatalog using the user's job name as the file name. If the $WAIT_QUEUE subcatalog does not exist at the time the job is terminated, it will be created for the user.

Remarks

- Jobs can be terminated if they are waiting to be initiated or have already been initiated.
- You only can terminate jobs for which you are the login user or the control user.
- A system operator can terminate any job.
- If a job is waiting to be initiated, it is eliminated as a candidate for execution.

- If a job 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.
- This command may have to be repeated several times. If after five attempts, the job does not terminate, the job will most likely not terminate until NOS/VE operations are suspended.
- For more information, see the NOS/VE System Usage manual.

Examples In this example, a previously submitted job named JOB_1 is terminated.

```
/terminate_job n=job_1
/disjs job_1
Name Not Found          :  job_1
```

TERMINATE_OUTPUT Command

Purpose Deletes a file (or files) from the NOS/VE output queue.

Format

TERMINATE_OUTPUT or
TERMINATE_OUTPUTS or
TERMINATE_PRINT or
TERMINATE_PRINTS or
TERO or
TERP
NAME=list of name
OUTPUT_STATE=keyword
STATUS=status variable

Parameters

NAME or NAMES or N

Specifies the output file(s) to be deleted. Values can be either the user-supplied or system-supplied name. This parameter is required.

OUTPUT_STATE or OS

Restricts the termination to output files in the specified state. Note that an initiated file will not be terminated until the output application has completed its processing of the file. Keywords are:

ALL

Output is terminated regardless of its state.

DEFERRED (D)

Restricts termination to files not yet eligible for printing.

QUEUED (Q)

Restricts termination to files waiting for printing.

INITIATED (I)

Restricts termination to files being printed.

Remarks

- To use this command you must be the login user, the control user, or a system operator.
- Specifying OUTPUT on the NAME parameter prevents output from being printed or written to the file specified by the OUTPUT_DISPOSITION parameter of the SUBMIT_JOB command.
- For more information, see the NOS/VE System Usage manual.

Examples

The following command deletes the output files named RESULTS and RESULTS_1 from the output queue:

```
/terminate_output name=(results,results_1)
```

TERMINATE_TASK Command

Purpose Terminates one or more asynchronous tasks. This command terminates the named tasks and all the tasks generated within those tasks. The specified tasks must have been initiated by the requesting task.

Format

TERMINATE_TASK or
TERT
TASK_NAME=list of name or **keyword**
STATUS=status variable

Parameters

TASK_NAME or **TASK_NAMES** or **TN**

Specifies the names of the tasks to be terminated. This may be the names you supply or the keyword ALL. If ALL is specified, all tasks initiated by the requesting task are terminated. This parameter is required.

Remarks For more information, see the NOS/VE System Usage manual.

$TIME Function

Purpose Returns the current time as a string.

Format

$TIME
(*keyword*)

Parameters *keyword*

Specifies the form in which the time is returned. Use one of the following keywords:

AMPM

Returns the time in terms of a 12-hour clock with an AM or PM designation, as in the following example:

```
6:38 PM
```

HMS

Returns the time in terms of a 24-hour clock, as in the following example:

```
18:38:14
```

The 24-hour designation for the hour is obtained by adding 12 to each hour after noon and treating midnight as 00:00:00. In the preceding example, the time is 14 seconds after 6:38 p.m.

ISOT

Returns the time as hours, minutes, seconds, and hundredths of a second, as in the following example:

```
18.38.14,79
```

ISOT stands for International Standards Organization Time.

MILLISECOND (MS)

Returns the time as hours, minutes, seconds, and thousandths of a second, as in the following example:

```
18:38:14.796
```

DEFAULT

Returns the default format of the time. This value is determined by your site.

If you omit this parameter, DEFAULT is used.

Remarks For further information about functions, see the NOS/VE System Usage manual.

Examples The following examples display the current date and time:

```
/display_value $time(ampm)//' on '//$date(month)
6:38 PM on March 28, 1987
```

TRANSFER_FILE_XMODEM Command

Purpose Initiates a file transfer between the mainframe and a microcomputer using Christensen protocol.

Format

TRANSFER_FILE_XMODEM or
TRAFX or
XMODEM
FILE_NAME=file
TRANSFER_DIRECTION=keyword
FILE_TYPE=keyword

LINE_FEEDS = boolean
SPECIAL_FILE_TYPE_BLOCKS = boolean
ERROR_CHECKING = keyword
FILE_MARKERS = keyword
CONFIGURATION_FILE = file
STATUS = status variable

Parameters *FILE_NAME* or *FN*

Specifies the file to be transferred. There is no default for this parameter. If you do not specify a file name the system prompts you for one.

TRANSFER_DIRECTION or *TD*

Indicates the direction of file transfer from the point of view of the host. There are two options available.

SEND (S)

File is sent from the host to a microcomputer.

RECEIVE (R)

File is received by the host from a microcomputer.

There is no default for this parameter. If you do not specify an option, the system prompts you for one.

FILE_TYPE or *FT*

Specifies the type of file received from a microcomputer. The following are valid keywords.

TEXT (T)

A text file is to be received.

BINARY (B)

A CYBER binary file is to be received.

OBJECT_LIBRARY (OL)

An object library is to be received.

MICRO_BINARY (MB)

A micro binary is to be received.

SELECT_AUTOMATICALLY (SA)

The file type to be received will be determined by the first block received.

VE_BACKUP

A NOS/VE backup file is to be received.

There is no default for this parameter.

If TRANSFER_DIRECTION=RECEIVE and you do not make an entry, you will be prompted for an entry.

If TRANSFER_DIRECTION=SEND, this parameter is ignored.

LINE_FEEDS or *LINE_FEED* or *LF*

Specifies whether a line feed is required after a carriage return to signal the end of a line.

If line 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, this block is never transmitted.

The default is FALSE.

This parameter is only used for transfers from the host (TRANSFER_DIRECTION=SEND).

When TRANSFER_DIRECTION=RECEIVE, this parameter is ignored and either a carriage return and a line feed or a carriage return alone will indicate the end-of-line.

SPECIAL_FILE_TYPE_BLOCKS or *SPECIAL_FILE_TYPE_BLOCK* or *SFTB* or *SP*

Specifies whether a nonstandard block is transmitted.

If TRUE, the nonstandard block is transmitted as the first block of the file.

If FALSE, this block is never transmitted.

The default is TRUE.

ERROR_CHECKING or *EC*

Determines whether error detection is done using checksums or a CRC algorithm. This parameter is only used when the host is the receiver (TRANSFER_DIRECTION=RECEIVE).

If TRANSFER_DIRECTION=SEND the microcomputer determines the error checking method used and this parameter is ignored. The following are valid keywords.

CRC (CR)

Error detection is done using the CCITT cyclic redundancy method.

CHECKSUM (CH)

Error detection is done using the checksum method.

If you do not specify a value, CRC is assumed.

FILE_MARKERS or *FILE_MARKER* or *FM*

Specifies both the sequences sent as file markers and the sequences received that are interpreted as file markers. This parameter is only applicable to text files.

You may specify one of the following keywords.

NOS (N) or NOSVE (N)

These keywords are equivalent. If the FILE_ MARKERS parameter is specified as NOS or NOSVE, 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 one of the sequences is received by the host alone on a line.

CPM (C) or MSDOS (M)

These keywords are equivalent. If the FILE_ MARKERS parameter is specified as CPM or MSDOS, 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, #EOP, and #EOI are recognized by the host as end-of-partition.

If you do not enter a value for this parameter, NOSVE is assumed.

CONFIGURATION_FILE or *CF*

Enables 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 $USER.PFTF_CONFIG does not exist, PFTF assumes there is no configuration file.

Remarks

- If you specify SEND on the TRANSFER_DIRECTION parameter, the file sent must not be empty and must have READ access.
- CONTROL X (the ASCII CAN character) will end or abort a transfer.
- If you specify RECEIVE on the TRANSFER_DIRECTION parameter and the file does not exist, it is created. If the file exists, it must have WRITE access. If the file is an old file (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.

- XMODEM cannot be used when you are connected to NOS/VE through INTERCOM.
- To use XMODEM, your data network must be able to set parity to NONE and pass 8-bit data.
- For more information, see the NOS/VE System Usage manual.

Examples The following example sends file MAIL from the host to a microcomputer.

```
/xmodem file_name=mail transfer_direction=send
```

In the next example you receive file TEST from a microcomputer.

```
/xmodem file_name=test transfer_direction=receive
```

You will be prompted for other parameters.

$TRANSLATE Function

Purpose Uses a translation table to change characters in a string.

Format

$TRANSLATE
(string or **keyword**
string)

Parameters

string or **keyword**

Specifies a keyword or string that specifes a translation table. This parameter is required. To use a NOS/VE-supplied translation table, specify one of the following keywords:

LOWER_TO_UPPER (LTU)

Produces a string in which all lowercase characters are changed to uppercase characters.

UPPER_TO_LOWER (UTL)

Produces a string in which all uppercase characters are changed to lowercase characters.

string

Specifies the string in which you want the characters changed. This parameter is required.

Remarks

- You can also create your own translation tables. Like the NOS/VE-supplied translation tables, a user-defined translation table is a string of 256 characters that uses the following algorithm:

```
result = ''
FOR i = 1 to $strlen($string) DO
  j = $ord($substr(string, i))
  result = result//$substr(table, j+1)
FOREND
```

- For further information about functions, see the NOS/VE System Usage manual.

Examples

- The following example uses the LOWER_TO_UPPER keyword to translate lowercase characters in a string to uppercase characters:

```
/display_value $translate(lower_to_upper,'123_abc')
123_ABC
```

- The following user-defined translation table converts every integer in a string to 9:

```
create_variable name=table kind=string value=''
FOR i=0 to 255 DO
  j=$char(i)
  IF ('0'<=j) and (j<='9') THEN
    j='9'
  IFEND
  table=table//j
  FOREND
```

The $TRANSLATE function is then used to change every integer to a 9:

```
/tt = $translate(table,'ab12xy3')
/display_variable tt
ab99xy9
```

$TRIM Function

Purpose Removes trailing space characters from a string.

Format

$TRIM
(string)

Parameters **string**

Specifies the string from which you want trailing space characters removed. This parameter is required.

Remarks For further information about functions, see the NOS/VE System Usage manual.

Examples Assume that file SAMPLE contains records with a 20-character NAME field followed by a 20-character ADDRESS field. To access the NAME field and delete any trailing space characters, enter the following commands:

```
/create_variable name=line kind=string
/accept_line input=sample variable=line
/name = $trim($substr(line,1,20))
```

$UNIQUE Function

Purpose Returns a 31-character name (beginning with a dollar sign) that is unique within the context of all NOS/VE systems.

Format **$UNIQUE**

Parameters None.

Remarks

- This function is used in procedures to create file names (via the $FNAME function) for scratch files.
- Since variable names cannot begin with a dollar sign, you must use a substring of $UNIQUE to create unique variable names.
- Do not use substrings of the value returned by the $UNIQUE function for other information, such as processor attibutes. Instead, use the values returned by the $PROCESSOR, $DATE, and $TIME functions.
- For further information about functions, see the NOS/VE System Usage manual.

Examples

- In the following example, a string representing a unique name is assigned to the variable SCRATCH_FILE. Subsequently, the file specified by the value contained in the INPUT parameter of the procedure is copied to file SCRATCH_FILE.

```
scratch_file = $unique
copy_file $value(input) $fname(scratch_file)
    .
    .   "Process scratch file
    .
```

- The following example illustrates the type of name the system returns:

```
/display_value $unique
$52389144OS0102D19870210T100435
```

UTILITY Command

Purpose

Delimits the definition and execution of a command utility.

Format

UTILITY
NAME=name
ENABLE_SUBCOMMAND_LOGGING=boolean
INTERACTIVE_INCLUDE_PROCESSOR=name or *keyword*
LIBRARY=list of file
LINE_PREPROCESSOR=name or *keyword*
ONLINE_MANUAL=name or *keyword*
PROMPT=string
SEARCH_MODE=keyword
TABLES=file
TERMINATION_COMMAND_NAME=name
STATUS=status variable

Parameters

NAME or **N**

Specifies the name used to refer to the utility while it is active. This parameter is required.

ENABLE_SUBCOMMAND_LOGGING or *ESL*

Determines whether the utility subcommands are logged (YES) or not (NO). Logging of commands occurs only for commands read either from an interactive file or the main

command file ($LOCAL.COMMAND) of a batch job. If this parameter is set to NO, the utility's subcommands are not logged, even if they are read from such a file.

If you omit this parameter, YES is assumed.

INTERACTIVE_INCLUDE_PROCESSOR or *IIP*

Reserved.

LIBRARY or *LIBRARIES* or *L*

Specifies the object library or libraries containing the processors for the utility subcommands. If you omit this parameter, the library containing the utility is used.

LINE_PREPROCESSOR or *LP*

Reserved.

ONLINE_MANUAL or *OM*

Specifies the online manual that describes the utility.This value is used as the default for the MANUAL parameter of a HELP command entered from within the utility.

Use the keyword NONE to designate that no online manual is associated with the utility.

If you omit this parameter, no online manual is associated with the utility.

PROMPT or *P*

Specifies the prompt string used for interactive command input. For command lines, a slash character (/) is appended to the specified prompt string. For command continuation lines, the string ../ is appended to the specified prompt string.

If you omit this parameter, the utility name is used.

SEARCH_MODE or *SM*

Specifies the command search mode used while processing commands for the utility. Values can be:

GLOBAL (G)

All entries in the command list can be searched. Commands specified by command name or file reference can be executed.

RESTRICTED (R)

All entries in the command list can be searched. In order for the search to proceed beyond the first entry in the command list, the command must be preceded by a slash (/). Commands specified by command name or file reference can also be executed.

EXCLUSIVE (E)

Only the entry at the beginning of the command list is searched for a command. Commands that are specified by command name or file reference are not allowed.

When you exit the utility, the command search mode is restored to the value it had when you entered the utility.

If you omit this parameter, GLOBAL is used.

TABLES or *TABLE* or *T*

Specifies the file containing the table of utility subcommands defined by the COMMAND command. This table is searched for subcommands while the utility is active.

If you omit this parameter, $COMMAND is used; that is, the file containing the UTILITY/UTILITYEND command is assumed to contain the command table.

TERMINATION_COMMAND_NAME or *TCN*

Specifies the utility subcommand used to terminate the utility. This must be one of the commands defined by the COMMAND command in the utility's command table.

If you omit this parameter, QUIT is assumed to be the termination command for the utility.

Remarks

- This command can only be executed from a procedure residing in an object library.
- Unlike many NOS/VE utilities, utilities you create using this command are not executed as a separate task within your job.
- For more information, see the NOS/VE System Usage manual.

$UTILITY Function

Purpose Returns the specified attribute of the currently executing command utility.

Format

$UTILITY
(keyword)

Parameters

keyword

Name of the attribute you want returned. This parameter is required. Use one of the following keywords:

NAME (N)

Returns the name of the currently active command utility. If no utility is currently active, NONE is returned.

ONLINE_MANUAL (OM)

Returns the name of the online manual associated with the utility. If no online manual is associated with the manual or no utility is active, NONE is returned.

PROMPT (P)

Returns the string the utility is using to prompt for interactive command input. If no utility is active, the null string is returned.

SUBCOMMAND_LOGGING_ENABLED (SCLE or SLE)

Returns a boolean value specifying whether subcommand logging is enabled (TRUE) or not (FALSE). If no utility is active, TRUE is returned.

Remarks For further information about functions, see the NOS/VE System Usage manual.

Examples

- The following example obtains the name of the active CREATE_OBJECT_LIBRARY utility:

```
COL/display_value  $utility(name)
CREATE_OBJECT_LIBRARY
COL/
```

- The following IF statement tests if the currently active utility is the SOURCE_CODE_UTILITY, which has the name SCU:

```
IF strrep($utility(name)) = 'SCU' THEN
... list of statements to be executed
IFEND
```

$VALIDATION_LEVEL Function

Purpose Returns the current validation level of the job.

Format **$VALIDATION_LEVEL**

Parameters None.

Remarks

- Returnable values are:

  USER

  During login, you must supply the system with valid family name, user name and password information.

  ACCOUNT

  During login, you must supply the system with a valid account number in addition to valid USER level information.

  PROJECT

  During login, you must supply the system with a valid project number in addition to valid USER and ACCOUNT level information.

  The values returned by this function are of type NAME.

- For more information on validation levels, see the NOS/VE System Usage manual.

$VALUE Function

Purpose Returns or sets the value of the specified parameter, or stores a value in a parameter from within the procedure.

Format

```
$VALUE
    (name
    integer1
    integer2
    keyword)
```

Parameters

name

Specifies the parameter for which you want a value returned, set, or stored. This parameter is required.

integer1

Specifies the number describing the position of the value set for which you want a value returned, set, or stored. The default value is 1.

Use this parameter if a parameter is defined as multiple value sets, each set having one value.

integer2

Specifies the number describing the position of the value element for which you want a value returned, set, or stored. The default value is 1.

Use this parameter if a parameter is defined as multiple value sets, each set having multiple value elements.

keyword

Specifies a range element for which you want a value returned; this parameter is valid only when the value is represented as a range. Use one of the following keywords:

LOW

Returns, sets, or stores the value of the low element. This is the default.

HIGH

Returns, sets, or stores the value of the high element.

Remarks

- This function is used to reference parameters within procedures.
- For further information about functions, see the NOS/VE System Usage manual.

Examples

- The following example is based on this procedure header:

```
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
    )
```

  Consider the following call to the preceding procedure:

```
/display_number (4,(2..3,2),5,6)
```

  The high element of the first value element in the second value set is displayed by including the following command in the DISPLAY_NUMBER procedure:

```
/display_value $value(number,2,1,high)
```

  The integer 3 is written to the output file.

- The next example shows how the $VALUE function can be used to set a variable kind parameter in a procedure. The following procedure has one parameter, ANSWER, which is a string variable. The procedure issues a prompt until a value starting with Y, y, N, or n is entered. The procedure then sets the value of the ANSWER parameter to Y or N.

```
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..
    (ltu input_answer) 1 1))
  UNTIL (input_answer = 'N')
  $value(answer) = input_answer

 PROCEND ask
```

An example of using this procedure follows:

```
/response=''
ask response
Enter Yes or No - you bet
/display_value response
Y
```

$VALUE_COUNT Function

Purpose Returns an integer count of the number of value elements supplied in a value set for a parameter.

Format

```
$VALUE_COUNT
  (name
  integer)
```

Parameters

name

Specifies the parameter for which you want to know the number of value elements in a value set. This parameter is required.

integer

Specifies the number describing the position of the value set for which you want to know the count of elements. The default value is 1.

Use this parameter if you expect multiple value sets (you can determine the number of value sets with the $SET_COUNT function).

Remarks

- This function is used to reference parameters within procedures.
- For further information about functions, see the NOS/VE System Usage manual.

Examples

The example is based on the following procedure header:

```
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
    )
```

Consider the following call to the preceding procedure:

```
/display_number ((7,8),4,(6,7))
```

In this case, the number of elements in the each value set is displayed by including the following commands in the procedure:

```
display_value $value_count(number,1)
```

The integer 2 is written to the output file.

```
display_value $value_count(number,2)
```

The integer 1 is written to the output file.

```
display_value $value_count(number,3)
```

The integer 3 is written to the output file.

$VALUE_KIND Function

Purpose

Returns a string specifying the type of value supplied for a parameter.

Format

$VALUE_KIND
(name
integer1
integer2
keyword)

Parameters

name

Specifies the parameter you are interrogating. This parameter is required.

integer1

Specifies the number describing the position of the value set for which you want to tknow the kind of value supplied. The default value is 1.

Use this parameter if a parameter is defined as multiple value sets, each set having one value.

integer2

Specifies the number describing the position of the value element for which you want to tknow the kind of value supplied. The default value is 1.

Use this parameter if a parameter is defined as multiple value sets, each set having multiple value elements.

keyword

Range element for which you want the value kind returned; this parameter is valid only when the value is represented as a range. Use one of the following keywords:

LOW

Returns the value kind of the low element. This is the default.

HIGH

Returns the value kind of the high element.

Remarks

- This function is used to reference parameters within procedures.
- This function is generally intended for parameters that can have a value kind of ANY or can be passed as keyword values.
- This function returns one of the following strings:

  'FILE'
  'NAME'
  'STRING'
  'INTEGER'
  'REAL'

'BOOLEAN'
'STATUS'
'VARIABLE'
'UNKNOWN'

If you do not specify this parameter, the string 'UNKNOWN' is returned.

If a parameter was defined as also accepting a keyword value, the string 'NAME' is returned.

If the parameter is specified with an application value, a string is returned that is defined by the application and describes the kind of value.

- For further information about functions, see the NOS/VE System Usage manual.

Examples

The example is based on the following procedure header:

```
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
    )
```

Consider the following call to the preceding procedure:

```
/display_number (1,(2..3)) out_file
```

In this case, the value kind of the second parameter is displayed by including the following command in the DISPLAY_NUMBER procedure:

```
/display_value ($value_kind(output)
```

File is written to the output file.

The value kind of the low range of the second value set for the NUMBER parameter is displayed by including the following command in the DISPLAY_NUMBER procedure:

```
/display_value ($value_kind(number,2,1,low)
```

INTEGER is written to the output file.

$VARIABLE
Function

Purpose Returns the attributes of a variable.

Format

```
$VARIABLE
    (name
    keyword)
```

Parameters

name

Specifies the variable for which you want the specified attribute returned. This parameter is required.

keyword

Specifies the variable attribute you want returned.

Chapter 3, Function Attributes, lists and describes the keyword values you can supply and the corresponding function results. This parameter is required.

Remarks

- The kind of result returned depends on the attribute being tested.
- When a string value is returned, all letters within the string are converted to uppercase.
- For further information about functions, see the NOS/VE System Usage manual.

Examples

The following example queries whether a name is currently a local variable. If it is, it is further queried for the kind of variable.

```
IF $variable(data_item,declared) = 'LOCAL' THEN
  IF $variable(data_item,kind) = 'BOOLEAN' THEN
    .
    .   "Process boolean variable.
    .
  ELSEIF $variable(data_item,kind) = 'INTEGER' THEN
    .
    .   "Process integer variable.
    .
  ELSEIF $variable(data_item,kind) = 'REAL' THEN
    .
    .   "Process real variable.
    .
  ELSEIF $variable(data_item,kind) = 'STATUS' THEN
    .
    .   "Process status variable.
    .
  ELSE
    .
    .   "Process string variable.
    .
  IFEND
ELSE
  create_variable data_item kind=integer
IFEND
```

VECTOR_FORTRAN Command

Purpose Calls and executes the FORTRAN Version 2 compiler.

Format

VECTOR_FORTRAN or
FTN2 or
VECF or
VFORTRAN or
VFTN or
FORTRAN2
INPUT=list of file
BINARY_OBJECT=file

LIST=file
COMPILATION_DIRECTIVES=boolean
DEBUG_AIDS=list of keyword
DEFAULT_COLLATION=keyword
ERROR=file
ERROR_LEVEL=keyword
EXPRESSION_EVALUATION=list of keyword
FORCED_SAVE=boolean
INPUT_SOURCE_MAP=list of file
LIST_OPTIONS=list of keyword
MACHINE_DEPENDENT=keyword
ONE_TRIP_DO=boolean
OPTIMIZATION_LEVEL=keyword
OPTIMIZATION_OPTIONS=list of keyword
REPORT_OPTIONS=keyword
RUNTIME_CHECKS=list of keyword
STANDARDS_DIAGNOSTICS=keyword
TARGET_MAINFRAME=keyword
TERMINATION_ERROR_LEVEL=keyword
VECTORIZATION_LEVEL=list of keyword
STATUS=status variable

Parameters

INPUT or *I*

Optional; specifies the name of the file or files containing the input source code. If more than one file is specified, the list of files is enclosed in parentheses. Omission causes $INPUT to be used.

BINARY_OBJECT or *BO* or *BINARY* or *B*

Optional; specifies the file to receive the binary object code produced. Omission selects $LOCAL.LGO. $NULL discards object code.

LIST or *L*

Optional; specifies the file to receive the compiler output listing. Omission selects $LIST for batch jobs and $NULL for interactive jobs.

COMPILATION_DIRECTIVES or *CD*

Optional; determines whether C$ directives within the source program are recognized. If ON is selected, the C$ directives are processed. OFF causes the C$ directives not to be processed. Omission selects ON.

DEBUG_AIDS or *DA*

Optional; selects debugging options. Options are:

PC

Generates argument checking information in the object code.

DT

Generates line number and symbol tables in the object code.

ALL

Selects both the PC and DT options.

NONE

Causes no argument checking information or line number and symbol tables to be generated in the object code.

Omission selects NONE.

DEFAULT_COLLATION or *DC*

Optional; specifies the weight table to be used for evaluating character expressions and by the CHAR and ICHAR functions. USER or U selects a user-specified weight table (DISPLAY). Omission, or FIXED or F, selects the fixed weight table (ASCII).

ERROR or *E*

Optional; specifies the name of the file to receive compiler-generated error messages. Omission writes the error messages to the file $ERRORS.

ERROR_LEVEL or *EL*

Optional; determines the minimum severity level for which errors are to be listed. All errors of severity greater than or equal to the specified level will be listed on the error and list files. Options are: TRIVIAL or T, INFORMATIONAL or I, WARNING or W, FATAL or F, or CATASTROPHIC or C. Omission selects WARNING.

EXPRESSION_EVALUATION or *EE*

Optional; controls the way the compiler evaluates expressions. Options are:

CANONICAL or C

Expressions are evaluated according to precedence rules.

MAINTAIN_EXCEPTIONS or ME

Inhibits optimizations that eliminate instructions that might cause runtime errors. Also causes floating-point comparisons for real or double precision operands.

MAINTAIN_PRECISION or MP

Inhibits optimizations that change a floating-point operation to a mathematically equivalent form.

OVERLAPPING_STRINGS_MOVES or OSM

Guarantees valid character assignment statements of the form v=exp where the character positions being defined in v are referenced in exp.

REFERENCE or R

Intrinsic functions are called by reference rather than by value. Also results in the generation of descriptive error messages by internal FORTRAN routines if execution errors occur. If this option is not selected, the operating system produces error messages which generally provide less information.

Omission, or NONE, selects no options. Multiple options are specified in the form (op, ..., op).

FORCED_SAVE or *FS*

Optional; specifies whether or not the values of variables and arrays in subprograms are to be retained after execution of a RETURN or END statement. ON saves variable and array values. This is equivalent to specifying a SAVE statement in every subprogram compiled. Omission, or OFF, does not save variable and array values.

INPUT_SOURCE_MAP or *ISM*

Specifies the file containing the source map that was generated by the OUTPUT_SOURCE_MAP option on the SCU EXPAND_DECK command. Omission causes $NULL to be used.

LIST_OPTIONS or *LO*

Optional; specifies the information that is to appear on the compiler output listing. Options are:

A

Lists the attributes of each symbolic name used or defined within the program.

M

A symbol attribute list (same as A option), Do loop map, and common block map are produced.

O

A listing of the generated object code is provided.

R

A cross reference listing; the listing shows the locations of the definition and use of each symbolic name in the program.

S

A listing of the program source statements is written to the output file.

SA

Same as the S option, except that lines turned off by C$ directives are listed.

NONE

No output listing is produced.

Omission selects the S option. Multiple options are specified in the form (op, ..., op).

MACHINE_DEPENDENT or *MD*

Optional; specifies whether the use of machine dependent capabilities within the program are to be diagnosed and if so, how severely:

NONE

Machine dependent usages are not diagnosed.

TRIVIAL (T) or INFORMATIONAL (I)

Machine dependent usages are diagnosed as trivial errors.

WARNING (W)

Machine dependent usages are diagnosed as warning errors.

FATAL (F)

Machine dependent usages are diagnosed as fatal errors, which result in a nonexecutable program.

Omission selects NONE.

ONE_TRIP_DO or *OTD*

Optional; sets the minimum trip count (mtc) for DO loops. The trip count is the number of times a DO loop is executed. ON sets the mtc to one. OFF sets the mtc to zero. Omission selects the OFF option.

If the terminal conditions of a DO loop are satisfied before the loop is entered, the mtc determines whether the loop is executed. If the mtc equals 1, the loop is executed once. If the mtc equals 0, the loop is not executed.

OPTIMIZATION_LEVEL or *OL* or *OPTIMIZATION* or *OPT*

Optional; selects the level of compiler optimization. Options are LOW, HIGH or DEBUG. DEBUG results in object code modified for use with Debug. Omission selects the LOW option.

OPTIMIZATION_OPTIONS or *OO*

Specifies instruction scheduling. This parameter is only significant when the OPTIMIZATION_LEVEL parameter specifies HIGH. Options are NONE and INSTRUCTION SCHEDULING. INSTRUCTION SCHEDULING allows for improved execution on the Model 990 regardless of the machine on which compilation occurs. NONE indicates

that instruction scheduling is determined by the values of the OPTIMIZATION_LEVEL and TARGET_MAINFRAME parameters as follows:

If TARGET_MAINFRAME specifies C180_CM or C180V and the compilation machine is a model 990, instruction scheduling occurs. If TARGET_MAINFRAME specifies C180_CM and the compilation machine is not a model 990, instruction scheduling also occurs. All other combinations of the TARGET_MAINFRAME and compilation machine do not produce instruction scheduling.

Omission selects NONE.

REPORT_OPTIONS or *RO*

Specifies the level of detail of the report messages on the listing file. BRIEF(B) selects brief mode messages. FULL(F) selects full mode messages. Omission selects NONE.

RUNTIME_CHECKS or *RC*

The RUNTIME_CHECKS parameter selects runtime range checking of subscripts and substrings. This parameter allows you to select multiple options. Options are:

Omitted

Same as RUNTIME_CHECK=NONE.

RUNTIME_CHECKS=NONE

Causes no options to be selected.

RUNTIME_CHECKS=R

Selects runtime range checking for character substring expressions. If a character substring expression would cause the substring to exceed the bounds declared by the CHARACTER statement, an informative diagnostic is issued and execution continues.

RUNTIME_CHECKS=S

Selects runtime range checking for subscript expressions. If a subscript expression would cause the substring to exceed its declared dimension bounds, an informative diagnostic is issued and execution continues.

RUNTIME_CHECKS=ALL

Selects both the R and S options.

NOTE

If RUNTIME_CHECKS specifies R, S, or ALL, and OPTIMIZATION_LEVEL=HIGH and VECTORIZATION_LEVEL=HIGH is also selected, the RUNTIME_CHECKS parameter is ignored.

STANDARDS_DIAGNOSTICS or *SD*

Optional; specifies whether or not non-ANSI source statements are to be diagnosed and, if so, how severely:

NONE

Nonstandard usages are not diagnosed.

TRIVIAL (T) or INFORMATIONAL (I)

Nonstandard usages are diagnosed as trivial errors.

WARNING (W)

Nonstandard usages are diagnosed as warning errors.

FATAL (F)

Nonstandard usages are diagnosed as fatal errors.

Omission selects NONE.

TARGET_MAINFRAME or *TM*

The TARGET_MAINFRAME parameter specifies the kind of mainframe for which the object code is generated. This parameter is only significant when the OPTIMIZATION_LEVEL parameter specifies HIGH. Options are:

Omitted

Same as TARGET_MAINFRAME=C180_CURRENT_MAINFRAME.

TARGET_MAINFRAME=C180_VECTOR
(TM=C180V)

The object code is generated for use on the model 990 of the CYBER 180. The model 990 has vector-processing capabilities.

TARGET_MAINFRAME=C180_MODEL_INDEPENDENT (TM=180MI)

The object code is generated for use on any model of the CYBER 180.

TARGET_MAINFRAME=C180_CURRENT_MAINFRAME (TM=C180CM)

The object code is generated for use on the machine on which compilation occurs.

For Better Performance

Be sure to use the TARGET_MAINFRAME=C180_VECTOR option for code that is going to be executed on a model 990 of the CYBER 180.

TERMINATION_ERROR_LEVEL or *TEL*

Optional; specifies the minimum error severity level for which the compiler is to return abnormal status. Options are: TRIVIAL(T), INFORMATIONAL(I), WARNING (W), FATAL (F) or CATASTROPHIC (C). Abnormal status is returned for all errors having severity equal or greater than the specified level. Omission selects FATAL.

VECTORIZATION_LEVEL or *VL* or *VECTORIZATION* or *VEC*

Specifies the vectorization level. NONE performs no vectorization. HIGH performs a high level of vectorization resulting in faster execution time but slower compilation time. Omission selects NONE.

Remarks

- For more information, see the FORTRAN Version 2 Language Definition manual.
- FORTRAN Version 2 is also known as VECTOR FORTRAN.

Examples

The following commands specify three parameters and select the default values for all other parameters:

```
/vector_fortran input=afile binary_object=bfile ..
../error_level=fatal

vftn i=afile b=bfile el=fatal
```

Options chosen:

INPUT=AFILE

Source statements are read from file AFILE

BINARY_OBJECT=BFILE

Object code is written to file BFILE

ERROR_LEVEL=FATAL

Only fatal and catastrophic errors are written to the error and list files

The following commands are equivalent; they select default values for all parameters except the INPUT parameter:

```
/vector_fortran input=myfile

          or

/vftn i=myfile
```

$VNAME Function

Purpose Converts a string to a variable name.

Format

```
$VNAME
   (string)
```

Parameters

string

Specifies the string you want converted to a variable name. This parameter is required.

Remarks

- This function makes it possible to reference a variable via a string.
- For further information about functions, see the NOS/VE System Usage manual.

Examples The following example creates an integer variable named COUNT and a string variable named COUNT_POINTER. To access the value of COUNT, the $VNAME function is given the value in COUNT_POINTER.

```
/count = 10
/count_pointer = 'count'
/display_value $vname(count_pointer)
10
```

VX Command

Purpose Places you in the VX/VE environment.

Format

VX
ARGUMENTS=string
GENERATE_LOAD_MAP=keyword
LIBRARIES=list of file
LOAD_MAP_OPTIONS=list of keyword
STATUS=status variable

Parameters

ARGUMENTS or *ARGS* or *A*

The first process executed by the process manager. Default is 'pm', which causes the regular login shell to be executed.

GENERATE_LOAD_MAP or *GLM*

Generate loadmaps for each task that is executed. The loadmap names are of the form "$LOCAL.$LOADMAPnn" where nn is the process slot number. The appropriate loadmap can be determined by looking in the job log for a message of the form "Generating $LOADMAPnn for pid=<process id>.".

If GENERATE_LOAD_MAP is omitted, a loadmap will not be generated.

LIBRARIES or *LIBRARY* or *L*

Replace the default bound emulation library with a list of user-specified libraries. The optional list of object libraries will be added to the beginning of the program library list. The pm searches the libraries in the order specified.

Maximum number of libraries that can be specified is ten.

WARNING: A bound emulation library must be one of the user-specified libraries.

This option is typically used to test a new version of the bound emulation library or to specify an ordered list of multiple libraries to be searched to satisfy externals for each process that pm starts.

If LIBRARY is omitted, the default VX/VE bound emulation library $SYSTEM.VX.LIB.BOUND_EM_LIB is used.

LOAD_MAP_OPTIONS or *LMO*

Set of one or more keywords indicating the information to be included in the load map. Options are:

NONE

No load map is written.

SEGMENT (S)

Segment map.

BLOCK (B)

Block map.

ENTRY_POINT (EP)

Entry point map.

CROSS_REFERENCE (CR)

Entry point cross-reference.

ALL

Segment map, block map, entry point map, and entry point cross-reference.

If LOAD_MAP_OPTION is omitted, the default load map options are not changed. The initial default option is NONE.

Remarks The VX/VE environment is documented in the VX/VE publications.

WAIT Command

Purpose Suspends command processing until either a specified number of milliseconds have elapsed, or another specified event has taken place.

Format

WAIT
TIME=integer
TASK_NAMES=list of name
QUEUE_NAMES=list of name
UNTIL=keyword
STATUS=status variable

Parameters

TIME or *T*

Specifies the number of milliseconds you want command processing suspended before the command sequence that issued the WAIT command is eligible to resume processing. If you omit this parameter, no suspension occurs.

TASK_NAMES or *TASK_NAME* or *TN*

Specifies the task(s) that must be completed before command processing can resume. The specified task(s) must have been initiated by the requesting task.

QUEUE_NAMES or *QUEUE_NAME* or *QN*

Specifies the job queue(s) from which a message must be received before command processing resumes.

UNTIL or *U*

Specifies whether any or all of a specified set of events must occur before the command terminates. This parameter can be immediately followed by one of the following keywords:

ANY

Stipulates that any of the specified events must occur. This is the default.

ALL

Stipulates that all of the specified events must occur.

Remarks

- This command affects only the task in which it was issued.
- For more information, see the NOS/VE System Usage manual.

Examples The following command causes a job to wait for 20,000 milliseconds (20 seconds).

```
/wait 20000
```

WAIT_FOR_SYSTEM_IDLE Command

Remarks Reserved for site personnel, Control Data, or future use.

WHEN Control Statement

Purpose Delimits a sequence of SCL statements that are to be executed when a specified condition occurs.

Format

WHEN condition names DO
statement list
WHENEND

Parameters

condition name

One or more names specifying conditions for which the sequence of statements is to be processed. Multiple condition names are separated by a comma or space. This parameter is required. The following are valid condition names:

PROGRAM_FAULT
LIMIT_FAULT
INTERRUPT
COMMAND_FAULT
ANY_FAULT

statement list

Specifies the statements that reside in the WHEN block.

Remarks

- You can use the CONTINUE statement to exit a WHEN block.
- The following variables are available for obtaining more information about a condition:

  OSV$STATUS

  Status variable initialized by the program that determined the condition.

  OSV$COMMAND_NAME

  String variable initialized as the name of the command being processed.

- For more information, see the NOS/VE System Usage manual.

Examples

The following is an example of establishing a condition handler. In this case, the statements following the WHEN clause are executed if a time limit condition is detected.

```
WHEN limit_fault DO
  put_line 'Incrementing time limit by 100 CP seconds.'
  change_job_limit name=cp_time ..
  resource_limit=($job_limit(cp_time, accumulator)+100)
  CONTINUE RETRY
WHENEND
```

WHILE Control Statement

Purpose

Provides for conditional repetition of a statement list.

Format

label: **WHILE**
boolean expression DO
statement list
WHILEND *label*

Parameters

label

Specifies the name of the WHILE block. This label can be used by CYCLE or EXIT statements within the block. The label on the WHILEND clause is optional and is used for documentation purposes only. It does not affect the meaning of the WHILEND statement.

boolean expression

Specifies the condition that must be TRUE for the following statement list to be executed. This parameter is required.

statement list

Specifies the statements that reside in the block.

Remarks

- The boolean expression is evaluated prior to each iteration of the statement list. If the expression is true, the statement list is executed. If the expression is FALSE, control passes to the statement following the WHILEND clause.
- For more information, see the NOS/VE System Usage manual.

Examples

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
    last_value = factorial_of * last_value
    factorial_of = factorial_of - 1
whilend factorial
/display_value last_value
120
/
```

Function Attributes 3

This appendix contains tables listing the attributes that can be returned for the $FILE, $JOB, $JOB_OUTPUT, $JOB_STATUS, $JOB_DEFAULT, $OUTPUT_STATUS, $PROGRAM, and $VARIABLE functions.

The following table shows the file attributes that can be returned by the $FILE function.

Table 3-1. File Attributes

Keyword	Description	Function Result
APPLICATION_ INFORMATION (AI)	Access control information used by an application. Refer to the CREATE_ FILE_PERMIT and CREATE_ CATALOG_PERMIT commands for more information.	String
ASSIGNED (A)	File within the requesting job is assigned to a device.	Boolean
ATTACHED	File is attached.	Boolean
CYCLE_NUMBER (CN)	Cycle number of a file.	Integer
DEVICE_CLASS (DC)	File is assigned to a NULL device.	'NULL'
	File is assigned to disk (mass storage).	'MASS_ STORAGE'
	File is assigned to a magnetic tape unit.	'MAGNETIC_TAPE'
	File is assigned to a terminal.	'TERMINAL'

(Continued)

Table 3-1. File Attributes *(Continued)*

Keyword	Description	Function Result
FILE_CONTENT (FC)	Content is unknown.	'UNKNOWN'
	Character data.	'LEGIBLE'
	Object module.	'OBJECT'
	List format (format effectors included).	'LIST'
	Screen formatting form definition.	'SCREEN'
FILE_LABEL_TYPE (FLT)	ANSI labels.	'LABELLED'
	Nonstandard labels.	'NON_ STANDARD_ LABELLED'
	No labels.	'UNLABELLED'
FILE_ORGANIZATION (FO)	File organization is sequential.	'SEQUENTIAL'
	File organization is byte-addressable.	'BYTE_ ADDRESSABLE'
	File organization is direct access.	'DIRECT_ ACCESS'
	File organization is indexed sequential.	'INDEXED_ SEQUENTIAL'

(Continued)

Table 3-1. File Attributes *(Continued)*

Keyword	Description	Function Result
FILE_PROCESSOR (FP)	Processor is unknown.	'UNKNOWN'
	ADA text.	'ADA'
	APL text.	'APL'
	NOS/VE ASSEMBLER text.	'ASSEMBLER'
	BASIC text.	'BASIC'
	C text.	'C'
	COBOL text.	'COBOL'
	CYBIL text.	'CYBIL'
	Debug object file.	'DEBUGGER'
	FORTRAN text.	'FORTRAN'
	LISP text.	'LISP'
	SCL text.	'SCL'
	SCU library.	'SCU'
	PASCAL text.	'PASCAL'
	PLI text.	'PLI'
	NOS PP ASSEMBLER text.	'PPU_ ASSEMBLER'
	PROLOG text.	'PROLOG'
	Associated with VX/VE.	'VX'

(Continued)

Table 3-1. File Attributes *(Continued)*

Keyword	Description	Function Result
FILE_STRUCTURE (FS)	Structure is unknown.	'UNKNOWN'
	Screen formatting file.	'FORM'
	Data file.	'DATA'
	Library file.	'LIBRARY'
GLOBAL_FILE_ POSITION (GFP)	File is positioned at beginning-of-information after the last file access request.	'BOI'
	File is positioned at beginning-of-partition after the last file access request.	'BOP'
	File is positioned at end-of-information after the last file access request.	'EOI'
	File is positioned at end-of-partition after the last file access request.	'EOP'
	File is positioned at end-of-record after the last file access request.	'EOR'
	File is positioned between the beginning and the end of a record after the last file access request.	'MID_RECORD'

(Continued)

Table 3-1. File Attributes *(Continued)*

Keyword	Description	Function Result
OPENED (O)	File is opened.	Boolean
OPEN_POSITION (OP)	File is positioned at beginning-of-information after an open operation.	'$BOI'
	File is positioned at end-of-information after an open operation.	'$EOI'
	File is not positioned after an open operation.	'$ASIS'
PERMANENT (P)	File is permanent.	Boolean
SIZE (S)	Length of the file in bytes.	Integer
TEMPORARY (T)	File is temporary.	Boolean
USER_INFORMATION (UI)	A string of up to 32 characters supplied by the user when the file was defined. Uppercase conversion is not performed.	String

The following table shows the job attributes that can be returned for the $JOB function.

Table 3-2. Job Attributes

Keyword	Description	Function Result
C170_OS_TYPE	The dual-state partner for the current job is NOS.	'NOS'
	The dual-state partner for the current job is NOS/BE.	'NOS/BE'
	There is no dual-state partner; the job is executing on a standalone system.	'NONE'
COMMENT_BANNER (CB)	Default character string displayed with output files generated by the job. Used as a comment about the printed output.	String
CONTROL_FAMILY (CF)	Family name of the control user. For most jobs, this is the family name of the login user.	Name
CONTROL_USER (CU)	Name of the control user. For most jobs, this is the name of the login user.	Name
COPIES (C)	Default number of copies to be made for output files generated by the job.	Integer

(Continued)

Table 3-2. Job Attributes *(Continued)*

Keyword	Description	Function Result
CYCLIC_AGING_ INTERVAL (CAI)	Number in milliseconds at which the working set of the job is aged.	Integer
DETACHED_JOB_ WAIT_TIME (DJWT)	Number of seconds the job remains suspended if detached or disconnected from the terminal session before the job terminates.	Integer or name
	The job is suspended indefinitelly.	UNLIMITED
DEVICE (D)	Default device name which, when combined with the station name, identifies a particular output device.	Name
DISPATCHING_ PRIORITY (DP)	Dispatching priority assigned as default to all user tasks.	Name
EXTERNAL_ CHARACTERISTICS (EC)	Default external characteristics string for output files generated by the job.	String
FORMS_CODE (FC)	Default forms code string for output files generated by the job.	String

(Continued)

Table 3-2. Job Attributes *(Continued)*

Keyword	Description	Function Result
JOB_ABORT_ DISPOSITION (JAD)	Action taken when a job is aborted due to a system failure:	Name
	Job is restarted.	RESTART
	Job is terminated.	TERMINATE
JOB_CLASS (JC)	Class of the current job.	Name
JOB_MODE (JM)	Mode of the current job.	name
JOB_MODE (JM)	Mode of the current job is batch.	BATCH
	Mode of the current job is interactive.	INTERACTIVE
	Interactive job was disconnected from a terminal as a result of a DETACH_JOB command.	INTERACTIVE_ COMMAND_ DISCONNECT
	Interactive job was disconnected from a terminal because of a problem with the communication line (such as hanging up a telephone).	INTERACTIVE_ LINE_ DISCONNECT
	Interactive job was disconnected from the terminal because of a system failure.	INTERACTIVE_ SYSTEM_ DISCONNECT

(Continued)

Table 3-2. Job Attributes *(Continued)*

Keyword	Description	Function Result
JOB_RECOVERY_ DISPOSITION (JRD)	Action taken following a system interrupt:	Name
	Job is continued after the point of interruption.	CONTINUE
	Job is restarted.	RESTART
	Job is terminated.	TERMINATE
JOB_SIZE (JS)	Size in bytes of the job input file. For interactive jobs, this value is always zero.	Integer
LOGIN_ACCOUNT (LA)	Account name under which the job is scheduled and run.	Name
LOGIN_FAMILY (LF)	Family name under which the job is scheduled and run.	Name
LOGIN_PROJECT (LP)	Project name under which the job is scheduled and run.	Name
LOGIN_USER (LU)	User name under which the job is scheduled and run.	Name
MAXIMUM_ WORKING_SET (MAXWS)	Maximum number of pages of memory allowed in the job's working set.	Integer or name
MINIMUM_ WORKING_SET (MINWS)	Minimum number of pages of memory allowed in the job's working set.	Integer

(Continued)

Table 3-2. Job Attributes *(Continued)*

Keyword	Description	Function Result
OPERATOR (O)	Indicates whether the current job has operator privileges.	Boolean
OPERATOR_FAMILY (OF)	Default family name of the private station operator who receives output files generated by the job.	Name
OPERATOR_USER (OU)	Default name of the private station operator who receives output files generated by the job.	Name
ORIGINATING_ APPLICATION_NAME (OAN)	Name of the application that caused the job to be entered into the system.	Name
OS_VERSION	Name and version of the operating system.	String
OUTPUT_CLASS (OC)	Default output class for output files generated by the job.	Name
OUTPUT_ DESTINATION (ODE)	Default identifier of system or station where output files are printed.	String

(Continued)

Table 3-2. Job Attributes *(Continued)*

Keyword	Description	Function Result
OUTPUT_ DESTINATION_USAGE (ODU)	Default use of the destination to which output files generated by the job are sent. One of the following activities takes place:	Name
	Output is sent to the NOS or NOS/BE system that shares the mainframe. If there is no dual state partner, output is queued indefinitely.	DUAL_STATE
	Output is sent via the network transfer facility to a remote system for printing.	NTF
	Output is sent to a private CDCNET I/O station.	PRIVATE
	Output is sent to a public CDCNET I/O station.	PUBLIC
	Output is sent via the queue file transfer facility to a remote system for printing.	QTF

(Continued)

Table 3-2. Job Attributes *(Continued)*

Keyword	Description	Function Result
OUTPUT_ DISPOSITION (ODI)	Specifies how the job's output is disposed.	Name or file
	Output is directed to the specified file.	File
	All output is discarded.	DISCARD_ALL_ OUTPUT
	Standard output is discarded.	DISCARD_ STANDARD_ OUTPUT
	Output generated by a job run on a remote system is directed to the printer on the remote system.	LOCAL
	Output is directed to the job owner's default output station.	PRINTER
	Output is directed to the job owner's $WAIT_QUEUE subcatalog.	WAIT_QUEUE
OUTPUT_PRIORITY (OP)	Default priority increment that is added to the initial output priority of output files generated by the job.	Name

(Continued)

Table 3-2. Job Attributes *(Continued)*

Keyword	Description	Function Result
PAGE_AGING_ INTERVAL (PAI)	Number of job mode central processing units (in milliseconds) at which the pages of memory in a job's working set are aged.	Integer
REMOTE_HOST_ DIRECTIVE (RHD)	Default directives for output files generated by the job if the OUTPUT_ DESTINATION_ USAGE parameter either is DUAL_ STATE or names a transfer facility.	String
ROUTING_BANNER (RB)	Default character string that is displayed with output files generated by the job.	String
SERVICE_CLASS (SC)	Service class of current job.	Name
SITE_INFORMATION (SI)	Character string that is established by the site when the job is queued.	String
STATION (S)	Default I/O station name to which output files generated by the job are sent.	Name

(Continued)

Table 3-2. **Job Attributes** *(Continued)*

Keyword	Description	Function Result
SWITCHn	Specifies whether the sense switch of the current job is on or off (n is an integer from 1 through 8).	Boolean
SYSTEM	Specifies whether the current job is the system job.	Boolean
SYSTEM_JOB_NAME (SJN)	Name assigned to the job by the system.	Name
USER_INFORMATION (UI)	Character string of 1 to 256 characters that is passed on to all output files generated by the job.	String
USER_JOB_NAME (UJN)	Name that was supplied by the user for the job.	Name
VERTICAL_PRINT_ DENSITY (VPD)	Default vertical print density at which the files generated by the job will be printed (SIX, EIGHT, or NONE).	Name
VFU_LOAD_ PROCEDURE (VLP)	Default name of the network procedure file which defines the vertical format unit (VFU) image to be loaded when a file generated by the job is printed.	Name

The following table shows the values that can be returned for the $JOB_DEFAULT function.

Table 3-3. Job Attribute Defaults

Keyword	Description	Function Result
CPU_TIME_LIMIT (CTL)	Maximum CPU time in seconds that is allocated for the job.	Integer
	This information is required during login.	REQUIRED
	No limit is set on the allocated CPU time for a job.	UNLIMITED
JOB_ABORT_ DISPOSITION (JAD)	Action taken when a job is aborted due to a system failure:	
	Job is restarted.	RESTART
	Job is terminated.	TERMINATE
JOB_CLASS (JC)	Default job class of the current job.	Name
JOB_RECOVERY_ DISPOSITION (JRD)	Action taken following a system interrupt.	Name
	Job is continued after the point of interruption.	CONTINUE
	Job is restarted.	RESTART
	Job is terminated.	TERMINATE

(Continued)

Table 3-3. Job Attribute Defaults *(Continued)*

Keyword	Description	Function Result
LOGIN_FAMILY (LF)	Family name under which the job is scheduled and run.	Name
MAGNETIC_TAPE_ LIMIT (MTL)	Maximum number of tapes drives that can be used simultaneously by the job.	Integer
	This information is required during login.	REQUIRED
	No limit is set on the number of tape drives simultaneously in use by the job.	UNLIMITED
	This parameter does not need to be specified in order for the job to be considered for a particular job class.	UNSPECIFIED
MAXIMUM_ WORKING_SET (MAXWS)	Maximum number of pages of memory allowed in the job's working set.	Integer
	This information is required during login.	REQUIRED
	The maximum number of pages allowed in the job's working set is unlimited.	UNLIMITED

(Continued)

Table 3-3. Job Attribute Defaults *(Continued)*

Keyword	Description	Function Result
OUTPUT_CLASS (OC)	Default output class for output files generated by the job.	Name
OUTPUT_ DESTINATION_USAGE (ODU)	Default use of the destination to which output files generated by the job are sent. One of the following activities takes place:	Name
	Output is sent to the NOS or NOS/BE system that shares the mainframe. If there is no dual state partner, output is queued indefinitely.	DUAL_STATE
	Output is sent via the network transfer facility to a remote system for printing.	NTF
	Output is sent to a private CDCNET I/O station.	PRIVATE
	Output is sent to a public CDCNET I/O station.	PUBLIC
	Output is sent via the queue file transfer facility to a remote system for printing.	QTF

(Continued)

Table 3-3. Job Attribute Defaults *(Continued)*

Keyword	Description	Function Result
SITE_INFORMATION (SI)	Character string that is established by the site when the job is queued.	String
SRU_LIMIT (SL)	Maximum system resource units allocated for the job.	Integer
	This information is required during login.	REQUIRED
	No limit is set on the allocated CPU time for a job.	UNLIMITED
STATION (S)	Default I/O station name to which output files generated by the job are sent.	Name
VERTICAL_PRINT_ DENSITY (VPD)	Default vertical print density at which the files generated by the job will be printed (SIX, EIGHT, or NONE).	Name

The following table shows the job_output attributes that can be returned for the $JOB_OUTPUT function.

Table 3-4. Job Output Attributes

Keyword	Description	Function Result
COMMENT_BANNER (CB)	Character string displayed with the output file; used as a comment about the output.	String
CONTROL_FAMILY (CF)	Family name of the control user of the output file.	Name
CONTROL_USER (CU)	Name of the control user of the output file.	Name
COPIES (C)	Number of copies to be printed.	Integer
COPIES_PRINTED (CP)	Number of copies already printed.	Integer
DATA_MODE (DM)	Type of data contained in the file. Either CODED or TRANSPARENT is returned.	Name
DEVICE (D)	Device name; when combined with the station name, this name identifies a particular output device.	Name
DEVICE_TYPE (DT)	Device type. For this release, this value is always PRINTER.	Name

(Continued)

Table 3-4. Job Output Attributes *(Continued)*

Keyword	Description	Function Result
EXTERNAL_ CHARACTERISTICS (EC)	External characteristics string for the output file.	String
FILE_POSITION (FP)	Point in the output file at which an interruption occurred. The restarting point is the beginning of information.	Integer
FILE_SIZE (FS)	Size of the output file in bytes.	Integer
FORMS_CODE (FC)	Forms code string for the output file.	String
LOGIN_ACCOUNT (LA)	Account name under which the job responsible for generating the output file was scheduled and run.	Name
LOGIN_FAMILY (LF)	Family name under which the job responsible for generating the output file was scheduled and run.	Name
LOGIN_PROJECT (LP)	Project name under which the job responsible for generating the output file was scheduled and run.	Name

(Continued)

Table 3-4. Job Output Attributes *(Continued)*

Keyword	Description	Function Result
LOGIN_USER (LU)	User name under which the job responsible for generating the output file was scheduled and run.	Name
OPERATOR_FAMILY (OF)	Family name of the private station operator who receives the output file.	Name
OPERATOR_USER (OU)	Name of the private station operator who receives the output file.	Name
ORIGINATING_ APPLICATION_NAME (OAN)	Name of the application that caused the job responsible for generating the output file to be entered into the system.	Name
OUTPUT_CLASS (OC)	Output class for the output file.	Name
OUTPUT_ DESTINATION (ODE)	Identifier of system or station where the output file is printed.	String

(Continued)

Table 3-4. Job Output Attributes *(Continued)*

Keyword	Description	
OUTPUT_ DESTINATION_USAGE (ODU)	Use of the destination to which the output file is sent. One of the following activities takes place:	
	Output is sent to the NOS or NOS/BE system that shares the mainframe. If there is no dual state partner, output is queued indefinitely.	DUAL_STATE
	Output is sent via the network transfer facility to a remote system for printing.	NTF
	Output is sent to a private CDCNET I/O station.	PRIVATE
	Output is sent to a public CDCNET I/O station.	PUBLIC
	Output is sent via the queue file transfer facility to a remote system for printing.	QTF
OUTPUT_PRIORITY (OP)	Priority increment that is added to the initial output priority of the output file.	Name

(Continued)

Table 3-4. Job Output Attributes *(Continued)*

Keyword	Description	Function Result
REMOTE_HOST_ DIRECTIVE (RHD)	Directives for the output file if the OUTPUT_ DESTINATION_ USAGE parameter either is DUAL_ STATE or names a transfer facility.	String
ROUTING_BANNER (RB)	Character string that is displayed with the output file.	String
SITE_INFORMATION (SI)	Character string that is established by the site when the job is queued.	String
STATION (S)	I/O station name to which the output file is to be sent (the control facility name if the OUTPUT_ DESTINATION_ USAGE attribute is PRIVATE).	Name
SYSTEM_FILE_NAME (SFN)	System-supplied name of the output file. This file name is generated by the NOS/VE system that executed the PRINT_FILE command.	Name
SYSTEM_JOB_NAME (SJN)	Name assigned by the system to the job responsible for generating the output file.	Name

(Continued)

Table 3-4. Job Output Attributes *(Continued)*

Keyword	Description	Function Result
USER_FILE_NAME (UFN)	User-supplied name of the output file. If no name is specified, the file name is used.	Name
USER_INFORMATION (UI)	Character string of 0 to 256 characters that are inherited from the job that generated the output file.	String
USER_JOB_NAME (UJN)	Name supplied by the user for the job responsible for generating the output file.	Name
VERTICAL_PRINT_ DENSITY (VPD)	Vertical print density at which the file will be printed (SIX, EIGHT, or NONE).	Name
VFU_LOAD_ PROCEDURE (VLP)	Name of the network procedure file which defines the VFU (vertical format unit) image to be loaded when the file is printed.	Name

The following table shows the job_status attributes that can be returned for the $JOB_STATUS function.

Table 3-5. Job Status Attributes

Keyword	Description	Function Result
CONTROL_FAMILY (CF)	Family name of the control user. For most jobs, this is the family name of login user.	Name
CONTROL_USER (CU)	Name of the control user. For most jobs, this is the name of login user.	Name
DISPLAY_MESSAGE (DM)	Displays either the last command submitted by the job or the last message returned to the job, whichever is most recent.	String
JOB_CLASS (JC)	Class of the specified job.	Name
JOB_DESTINATION_ USAGE (JDU)	Specifies the queue file transfer application used to forward the job to a remote system for execution.	Name
JOB_MODE (JM)	Mode of the specified job is batch.	'BATCH'
	Mode of the specified job is interactive.	'INTERACTIVE'

(Continued)

Table 3-5. Job Status Attributes *(Continued)*

Keyword	Description	Function Result
	Specified job was disconnected from a terminal as a result of a DETACH_JOB command.	'INTERACTIVE_ COMMAND_ DISCONNECT'
	Specified job was disconnected from a terminal because of a problem with the communication line (such as hanging up a telephone).	'INTERACTIVE_ LINE_ DISCONNECT'
	Interactive job was disconnected from the terminal because of a system failure.	'INTERACTIVE_ SYSTEM_ DISCONNECT'
JOB_STATE (JS)	Returns the state of a job.	
	Job has finished executing.	COMPLETED
	Job is not yet eligible to be initiated.	DEFERRED
	Job has been initiated.	INITIATED
	Job is waiting to be initiated.	QUEUED
	Job has been terminated.	TERMINATED
	The specified job is not known to the system.	UNKNOWN

(Continued)

Table 3-5. Job Status Attributes *(Continued)*

Keyword	Description	Function Result
LOGIN_FAMILY (LF)	Family name under which the job is scheduled and run.	Name
LOGIN_USER (LU)	User name under which the job is scheduled and run.	Name
OPERATOR_ACTION_ POSTED (OAP)	Indicates whether the job is waiting for operator action.	Boolean
SYSTEM_JOB_NAME (SJN)	Name assigned to the job by the system.	Name
USER_JOB_NAME (UJN)	Name that was supplied by the user for the job.	Name

The following table shows the output_status attributes that can be returned for the $OUTPUT_STATUS function.

Table 3-6. Output Status Attributes

Keyword	Description	Function Result
CONTROL_FAMILY (CF)	Family name of the control user of the output file.	Name
CONTROL_USER (CU)	Name of the control user of the output file.	Name
LOGIN_FAMILY (LF)	Family name of the job that generated the output file.	Name
LOGIN_USER (LU)	User name of the job that generated the output file.	Name
OUTPUT_ DESTINATION_USAGE (ODU)	Use of the destination to which the output file is sent. One of the following activities takes place:	
	Output is sent to the NOS or NOS/BE system that shares the mainframe.	DUAL_STATE
	Output is sent via the network transfer facility to a remote system for printing.	NTF
	Output is sent to a private CDCNET I/O station.	PRIVATE

(Continued)

Table 3-6. Output Status Attributes *(Continued)*

Keyword	Description	Function Result
	Output is sent to a public CDCNET I/O station.	PUBLIC
	Output is sent via the queue file transfer facility to a remote system for printing.	QTF
OUTPUT_STATE (OS)	Returns the status of the output file.	
	File is being printed.	INITIATED
	File is waiting to be printed.	QUEUED
	Printing process for the file has been terminated.	TERMINATED
	The specified output file is not known to the system.	UNKNOWN
SYSTEM_FILE_NAME (SFN)	System-supplied name of the output file. This file name is generated by the NOS/VE system that executes the PRINT_FILE command.	Name

(Continued)

Table 3-6. Output Status Attributes *(Continued)*

Keyword	Description	Function Result
SYSTEM_JOB_NAME (SJN)	Name assigned by the system to the job responsible for generating the output file.	Name
USER_FILE_NAME (UFN)	User-supplied name of the output file. If no name is specified, the file name is used.	Name

The following table shows the program attributes that can be returned for the $PROGRAM function.

Table 3-7. Program Attributes

Keyword	Description	Function Result
ABORT_FILE (AF)	Abort file.	File
ARITHMETIC_ OVERFLOW (AF)	Indicates whether an interrupt will occur when an ARITHMETIC_ OVERFLOW condition is encountered.	Boolean
ARITHMETIC_LOSS_ OF_SIGNIFICANCE (ALOS)	Indicates whether an interrupt will occur when an ARITHMETIC_ LOSS_OF_ SIGNIFICANCE condition is encountered.	Boolean
DEBUG_INPUT (DI)	Debug subcommand file.	File
DEBUG_MODE (DM)	Indicates whether debug mode is on.	Boolean
DEBUG_OUTPUT (DO)	Debug output file.	File
DIVIDE_FAULT (DF)	Indicates whether an interrupt will occur when a DIVIDE_ FAULT condition is encountered.	Boolean

(Continued)

Table 3-7. Program Attributes *(Continued)*

Keyword	Description	Function Result
EXPONENT_ OVERFLOW (EO)	Indicates whether an interrupt will occur when an EXPONENT_ OVERFLOW condition is encountered.	Boolean
EXPONENT_ UNDERFLOW (EU)	Indicates whether an interrupt will occur when an EXPONENT_ UNDERFLOW condition is encountered.	Boolean
FP_INDEFINITE (FPI)	Indicates whether an interrupt will occur when a FLOATING_ POINT_INDEFINITE condition is encountered (FP stands for *floating point*).	Boolean
FP_LOSS_OF_ SIGNIFICANCE (FPLOS)	Indicates whether an interrupt will occur when a FLOATING_ POINT_LOSS_OF_ SIGNIFICANCE condition is encountered (FP stands for *floating point*).	Boolean

(Continued)

Table 3-7. Program Attributes *(Continued)*

Keyword	Description	Function Result
INVALID_BDP_DATA (IBDPD)	Indicates whether an interrupt will occur when a FLOATING_POINT_LOSS_OF_SIGNIFICANCE condition is encountered (BDP stands for *business data processing*).	Boolean
LOAD_MAP (LM)	Load map file.	File
LOAD_MAP_OPTION (LMO)	Indicates whether load map option specified by keyword value is selected. If a LOAD_MAP_OPTION program attribute is specified, one of the keywords listed in the Remarks section must also be supplied.	Boolean
PRESET_VALUE (PV)	Preset value is 0.	'ZERO'
	Preset value is floating point indefinite.	'FLOATING_POINT_INDEFINITE'
	Preset value is infinity.	'INFINITY'
	Preset value is alternate 1's.	'ALTERNATE_ONES'

(Continued)

Table 3-7. Program Attributes *(Continued)*

Keyword	Description	Function Result
TERMINATION_ ERROR_LEVEL (TEL)	Termination error level is warning.	'WARNING'
	Termination error level is error.	'ERROR'
	Termination error level is fatal.	'FATAL'

The following table shows the variable attributes that can be returned for the $VARIABLE function.

Table 3-8. Variable Attributes

Keyword	Description	Function Result
DECLARED	Variable is not accessible in the current block.	'UNKNOWN'
	Variable is declared in the current block.	'LOCAL'
	Variable is declared in an outer block with an externally declared (XDCL) scope.	'NONLOCAL'
KIND	Boolean variable.	'BOOLEAN'
	Integer variable.	'INTEGER'
	Real variable.	'REAL'
	Status variable.	'STATUS'
	String variable.	'STRING'
LOWER_BOUND	Lower bound (smallest subscript) of an array variable (1 if the variable is not an array).	Integer

(Continued)

Table 3-8. Variable Attributes *(Continued)*

Keyword	Description	Function Result
STRING_SIZE	Maximum number of characters that can be held by a string variable (if the variable is not a string variable, an abnormal status is returned).	Integer
UPPER_BOUND	Upper bound (largest subscript) of an array variable (1 if the variable is not an array).	Integer

ADMINISTER_RECOVERY_LOG 4

ADMINISTER_RECOVERY_LOG Command

Purpose Begins an Administer_Recovery_Log utility session.

Format

ADMINISTER_RECOVERY_LOG or
ADMRL
STATUS=status variable

Remarks For more information, see the NOS/VE Advanced File Management manual.

Examples The following is the minimal Administer_Recovery_Log session; it does nothing.

```
/administer_recovery_log
admrl/quit
```

To see a list of available subcommands you can type HELP while in this utility.

BACKUP_LOG ADMRL Subcommand

Purpose Initiates an immediate backup of the log.

Format

BACKUP_LOG or
BACL
STATUS=status variable

Remarks

- This subcommand must be preceded in the session by a USE_LOG subcommand to specify the log to be backed up.
- This subcommand can be performed only on a log that has been configured for log backups. (This is done using the CONFIGURE_LOG_BACKUP subcommand.)

- You should use the BACKUP_LOG subcommand in both of the following situations:
  - Log users are receiving the status AAE$LOG_TEMPORARILY_FULL, which indicates that an immediate repository switch is needed.
  - A system failure seems imminent.
- For more information see the NOS/VE Advanced File Management Usage manual.

Examples The following session initiates an immediate repository switch and backup for the existing log in $USER.MY_LOG.

```
/administer_recovery_log
admrl/use_log,catalog=$user.my_log
admrl/backup_log
admrl/quit
/
```

CANCEL_LOG_CHANGES ADMRL Subcommand

Purpose Discards the log specifications and any delete requests accumulated in the session.

Format

CANCEL_LOG_CHANGES or
CANLC
STATUS=status variable

Remarks

- This subcommand discards the accumulated log specifications and delete requests before they are put into effect by the QUIT subcommand.
- The CANCEL_LOG_CHANGES subcommand is appropriate only after a USE_LOG subcommand has been entered.
- You can begin accumulating log specifications again after the CANCEL_LOG_CHANGES subcommand. To do so, you must begin with another USE_LOG subcommand to specify the log to be created or changed.

- For more information, see the NOS/VE Advanced File Management Usage manual.

Examples The following session enters a change for $USER.MY_LOG, but then discards the change so the session does nothing.

```
/administer_recovery_log
admrl/use_log, $user.my_log, ..
admrl../set_performance_option, emphasis=speed
admrl/cancel_log_changes
admrl/quit
/
```

CLEAR_PROBLEM_JOURNAL ADMRL Subcommand

Purpose Clears the problem journal for the log.

Format **CLEAR_PROBLEM_JOURNAL** or **CLEPJ**

STATUS=status variable

Remarks

- The system maintains a problem journal in each log in which it records any problems that occur while using the log.
- You must display the problem journal before clearing it. To do so, use the DISPLAY_PROBLEM_JOURNAL subcommand.
- The log referenced by a CLEAR_PROBLEM_JOURNAL subcommand is the log specified on the USE_LOG subcommand earlier in the session.
- For more information, see the NOS/VE Advanced File Management Usage manual.

Examples The following session prints the contents of the problem journal for $USER.MY_LOG before clearing the problem journal.

```
/administer_recovery_log
admrl/use_log, $user.my_log
admrl/display_problem_journal, output=log_problems
admrl/print_file, log_problems
admrl/clear_problem_journal
admrl/quit
/
```

CONFIGURE_LOG_BACKUP ADMRL Subcommand

Purpose Establishes the backup file pool for the log.

Format

CONFIGURE_LOG_BACKUP or
CONLB
ADD_FILE=file
REMOVE_FILE=file
MEDIA=keyword
EXTERNAL_VSN=list of string
RECORDED_VSN=list of string
TYPE=keyword
VERIFY=boolean
STATUS=status variable

Parameters *ADD_FILE* or *AF*

File to be added to the pool of backup files for the log. If ADD_FILE is omitted, no backup file is added.

REMOVE_FILE or *RF*

File to be removed from the pool of backup files for the log. If REMOVE_FILE is omitted, no backup file is removed.

MEDIA or *M*

Device class of the file specified by the ADD_FILE parameter.

MAGNETIC_TAPE_DEVICE or MTD

Indicates that the log files are backed up to a labeled tape.

MASS_STORAGE_DEVICE or MSD

Indicates that the log files are backed up to disk. (The next four parameters are not used.)

The default value is MAGNETIC_TAPE_DEVICE.

EXTERNAL_VSN or *EVSN*

List of external VSNs identifying the tape volumes that compose the file specified by the ADD_FILE parameter. The VSNs are specified as strings of from 1 through 6 characters enclosed in apostrophes. This parameter must be specified if MEDIA is set to MAGNETIC_TAPE_DEVICE.

RECORDED_VSN or *RVSN*

List of recorded VSNs of the tape volumes that compose the file specified by the ADD_FILE parameter. The recorded VSN is in the ANSI VOL1 label on the volume. The VSNs are specified as strings of from 1 through 6 characters enclosed in apostrophes. This parameter must be specified if MEDIA is set to MAGNETIC_TAPE_DEVICE.

TYPE or *T*

Tape density written by a nine-track tape drive for the file specified by the ADD_FILE parameter. This parameter is used only if MEDIA is set to MAGNETIC_TAPE_DEVICE.

MT9$800

800 cpi.

MT9$1600

1600 cpi.

MT9$6250

6250 cpi.

The default value is MT9$6250.

VERIFY or *V*

Indicates whether the backup file specified by the ADD_FILE parameter is verified. This parameter is used only if MEDIA is set to MAGNETIC_TAPE_DEVICE.

TRUE or YES or ON

The magnetic tape is mounted; the backup file is opened to verify that it exists and that it has read and write capabilities.

FALSE or NO or OFF

The backup file is not verified.

The default value is TRUE.

Remarks

- A mass storage backup file is specified by its file path. However, any file cycle specification on the file path is ignored. The backup is always written to cycle 1. (Cycle 1 is created if it does not exist and overwritten if it does exist.)
- If any backup files are configured for the log, a backup file must be configured for each log repository. For example, if backup files are configured, a log with five repositories must have five backup files.
- The FILE_CLASS and INITIAL_VOLUME parameters are described in detail as parameters of the REQUEST_MASS_STORAGE command in the NOS/VE System Performance and Maintenance, Volume 2, Maintenance manual.
- For more information, see the NOS/VE Advanced File Management Usage manual.

CONFIGURE_LOG_RESIDENCE ADMRL Subcommand

Purpose Establishes configuration of the log.

Format

CONFIGURE_LOG_RESIDENCE or
CONLR
REPOSITORIES=integer
REPOSITORY_SWITCHING_SIZE=integer
REPOSITORY_SWITCHING_TIME=integer
SWITCH_SUPPRESSION_SIZE=integer or *keyword*
SWITCH_SUPPRESSION_TIME=integer or *keyword*
REPOSITORY_SIZE_LIMIT=integer
STATUS=status variable

Parameters

REPOSITORIES or *R*

Number of disk-resident repositories for the log (integer from 2 through 4096). The default value is 5.

If a backup account or backup pool is specified for the log, the log must have at least 3 repositories.

REPOSITORY_SWITCHING_SIZE or *RSS*

Repository size threshold for the log (in bytes, from 500,000 through 2,132,483,647 [$(2^{31} - 1) - 15,000,000$]). The default value is 70,000,000 bytes.

REPOSITORY_SWITCHING_TIME or *RST*

Repository time threshold for the log (in minutes, from 1 through 525,600 [365 days]). The default value is 1440 (24 hours).

SWITCH_SUPPRESSION_SIZE or *SSS*

Minimum repository size required before switching (in bytes, from 500,000 through 2,132,483,647 [$(2^{31} - 1) - 15,000,000$]). The default value is 0.

SWITCH_SUPPRESSION_TIME or *SST*

Minimum repository time required before switching (in minutes, from 1 through 525,600 [365 days]). The default value is 0.

REPOSITORY_SIZE_LIMIT or *RSL*

Absolute maximum repository size limit (in bytes, from 15,500,000 through 2,147,483,647 [$2^{31} - 1$]). It must be at least 15,000,000 bytes larger than the REPOSITORY_SWITCHING_SIZE. The default value is 100,000,000 bytes.

Remarks

- You cannot modify an existing log while any keyed file that uses the log is being updated. The subcommand notifies you when it cannot get exclusive access to the log. You should then quit the session and try again later.

- This subcommand can be specified only for a new log. The configuration cannot be changed for an existing log.

- During normal log activity, the active repository size should never approach the REPOSITORY_SIZE_LIMIT.
- The FILE_CLASS and INITIAL_VOLUME parameters are described in detail as parameters of the REQUEST_MASS_STORAGE command in the NOS/VE System Performance and Maintnenance, Volume 2, Maintenance manual.
- For more information see the NOS/VE Advanced File Management Usage manual.

DELETE_LOG ADMRL Subcommand

Purpose Requests deletion of an existing log.

Format

DELETE_LOG or
DELL
CATALOG=file
RETAIN_CONFIGURATION=boolean
STATUS=status variable

Parameters

CATALOG or **C**

Catalog path of the log to be deleted. This parameter is required.

RETAIN_CONFIGURATION or **RC**

Indicates whether the log configuration is kept.

TRUE or YES or ON

Empty the repositories and the log journal, but keep the log configuration.

FALSE or NO or OFF

Delete all files composing the log, including the repositories, the log journal, and mass storage log backup files.

This parameter is required.

Remarks

- The logs specified by DELETE_LOG subcommands are not deleted until the QUIT subcommand is entered for the session. A CANCEL_LOG_CHANGES subcommand clears any pending deletion requests.
- If the log configuration is to be retained, the subcommand deletes all the log data, but the log data on the repositories continues to exist and can continue to be used.

  If the log configuration is not to be retained, the subcommand requests deletion of all files relating to the log in the catalog. The catalog will no longer be usable as a log until a new log is created in it.

  If the subcommand requests deletion of all files in the catalog, the catalog is deleted as well.
- The catalog used is specified on the DELETE_LOG subcommand. Therefore, the subcommand does not reference the log specified by the USE_LOG subcommand. More than one log can be deleted in a session.
- For more information see the NOS/VE Advanced File Management Usage manual.

Examples The following session requests deletion of log $USER.MY_LOG, but then cancels the request:

```
/administer_recovery_log
admrl/delete_log, $user.my_log, retain_configuration=false
admrl/cancel_log_changes
admrl/quit
/
```

DISPLAY_LOG_CONFIGURATION ADMRL Subcommand

Purpose Displays the current log specifications.

Format **DISPLAY_LOG_CONFIGURATION** or **DISLC**

OUTPUT=file
STATUS=status variable

Parameters *OUTPUT* or *O*

File to which the display is written.

The subcommand positions the file according to the file position ($BOI, $EOI) appended to the file reference or, if no position is specified, according to its OPEN_POSITION attribute value.

If OUTPUT is omitted, the display is written to the standard output file, $OUTPUT.

Remarks

- This subcommand must be preceded in the session by a USE_LOG subcommand to specify the log whose configuration is displayed.
- For more information see the NOS/VE Advanced File Management Usage manual.

DISPLAY_PROBLEM_JOURNAL ADMRL Subcommand

Purpose Displays the problem journal for the log.

Format

DISPLAY_PROBLEM_JOURNAL or
DISPJ
OUTPUT=file
STATUS=status variable

Parameters *OUTPUT* or *O*

File to which the display is written.

The subcommand positions the file according to the file position ($BOI, $EOI) appended to the file reference or, if no position is specified, according to its OPEN_POSITION attribute value.

If OUTPUT is omitted, the display is written to the standard output file, $OUTPUT.

Remarks

- The system records any problems that have occurred while using the log in the problem journal for the log.
- The log referenced by a DISPLAY_PROBLEM_JOURNAL subcommand is the log specified on the USE_LOG subcommand earlier in the session.
- For more information see the NOS/VE Advanced File Management Usage manual.

Examples The following session writes the problem journal for $USER.MY_LOG to file LOG_PROBLEMS and prints it.

```
/administer_recovery_log
admrl/use_log, $user.my_log
admrl/display_problem_journal, ..
admrl../output=log_problems
admrl/print_file, log_problems
admrl/quit
/
```

HELP
ADMRL Subcommand

Purpose Provides access to online information about the utility.

Format

HELP or
HEL
SUBJECT=string
MANUAL=file
STATUS=status variable

Parameters

SUBJECT or *S*

Topic to be found in the index of the online manual. The topic must be enclosed in apostrophes ('topic').

If you omit the SUBJECT parameter, HELP displays a list of the available subcommands and prompts for display of a subcommand description in the online manual.

MANUAL or *M*

Online manual file to be read. If you omit the MANUAL parameter, the default is AFM. The subcommand searches for the file in the working catalog and then in the $SYSTEM.MANUALS catalog.

Remarks

- If the SUBJECT parameter specifies a topic that is not in the manual index, a nonfatal error is returned notifying you that the topic could not be found.
- The default manual file, $SYSTEM.MANUALS.AFM, contains the online version of the NOS/VE Advanced File Management Usage manual, as provided with the NOS/VE system.

- If your terminal is defined for screen applications, online manuals are displayed in screen mode. Help is available for reading the online. To leave the online manual and return to the utility, use QUIT.
- For more information, see the NOS/VE Advanced File Management Usage manual.

Examples The following session shows the default display returned by the HELP subcommand.

```
/administer_recovery_log
admrl/help
The following Administer_Recovery_Log subcommands are available:
BACKUP_LOG
CANCEL_LOG_CHANGES
CLEAR_PROBLEM_JOURNAL
CONFIGURE_LOG_BACKUP
CONFIGURE_LOG_RESIDENCE
DELETE_LOG
DISPLAY_LOG_CONFIGURATION
DISPLAY_PROBLEM_JOURNAL
HELP
QUIT
SET_LOG_BACKUP_ACCOUNT
SET_PERFORMANCE_OPTION
SET_VERIFICATION_LEVEL
```

For a description of a subcommand in the online manual, enter: HELP subject = '<subcommand>'

To return from an online manual, enter: QUIT

```
admrl/quit
/
```

QUIT
ADMRL Subcommand

Purpose Executes the accumulated log specifications and ends the session.

Format

QUIT or
QUI
APPLY_LOG_CHANGES=boolean
STATUS=status variable

Parameters *APPLY_LOG_CHANGES* or *ALC*

Indicates whether the log repositories are created or updated based upon the accumulated log specifications.

TRUE or YES or ON

The log is created or updated. Any logs specified on a DELETE_LOG subcommand during the session are deleted.

If a new log is being created, the log catalog is created if it does not exist. The log files are created and initialized. If the log catalog already exists, only the performance option and backup account information can be changed.

FALSE or NO or OFF

Log repositories are not created or updated; log specifications are discarded. Any logs specified on a DELETE_LOG subcommand during the session are kept.

The default value is TRUE.

Remarks

- To discard the accumulated log specifications or delete requests before ending the session, enter a CANCEL_LOG_CHANGES subcommand before entering the QUIT subcommand.
- The changes specified by the following subcommands do not take effect until the log changes are applied when the QUIT subcommand is entered:

  CONFIGURE_LOG_BACKUP
  CONFIGURE_LOG_RESIDENCE
  DELETE_LOG
  SET_LOG_BACKUP_ACCOUNT
  SET_PERFORMANCE_OPTION
  SET_VERIFICATION_LEVEL

- For more information, see the NOS/VE Advanced File Management Usage manual.

SET_LOG_BACKUP_ACCOUNT ADMRL Subcommand

Purpose Specifies the validation information used by backup jobs for the log.

NOTE

Each time the password is changed for the user name used as the backup account, the password must also be changed in the log configuration. Otherwise, all subsequent backup jobs fail to execute.

Format

SET_LOG_BACKUP_ACCOUNT or
SETLBA
USER=name
PASSWORD=name
FAMILY_NAME=name
USER_JOB_NAME=name
JOB_CLASS=name
ACCOUNT=name
PROJECT=name
OUTPUT_DISPOSITION=file or *keyword*
USER_INFORMATION=string
STATUS=status variable

Parameters

USER or **U**

User name under which backup jobs are run. This parameter is required.

PASSWORD or **PW**

Password for the user name specified by the USER parameter. This parameter is required.

FAMILY_NAME or *FN*

Optional family name under which backup jobs are run. If FAMILY_NAME is omitted, backup jobs run under the family to which the specified user name belongs.

USER_JOB_NAME or *JOB_NAME* or *UJN* or *JN*

Optional name by which the backup jobs are identified in the system. If USER_JOB_NAME is omitted, the name assigned backup jobs is the user name.

JOB_CLASS or *JC*

Optional job class in which the backup jobs are run. If JOB_CLASS is omitted, the jobs run in the default job class for the user name.

ACCOUNT or *A*

Account to which resource usage is charged for the backup jobs. If you omit this parameter for a user name that requires an account, the backup jobs will fail to execute. (See the Remarks.)

PROJECT or *P*

Project to which resource usage is charged for the backup jobs. If you omit this parameter for a user name that requires a project, the backup jobs will fail to execute. (See the Remarks.)

OUTPUT_DISPOSITION or *OD* or *ODI* or *STANDARD_OUTPUT* or *SO*

Specifies the default for how the backup job's standard output is to be disposed. If omitted, the attribute associated with this parameter does not change.

File name

The standard output is copied to the specified file name at job end.

DISCARD_ALL_OUTPUT or DAO

All output generated by the backup job is to be discarded at job end.

DISCARD_STANDARD_OUTPUT or DSO

Standard output is to be discarded at job end.

LOCAL or L

Any output generated by the backup job is printed at the destination system rather than being returned to the originating user's default output station.

PRINTER or P

Any output generated by the backup job is returned to the originating user's default output station.

WAIT_QUEUE or WQ

Any output generated by the backup job is returned to the originating user's $WAIT_QUEUE subcatalog on the originating system using the user's job name for the file name. If the $WAIT_QUEUE subcatalog does not exist at the time the output files are returned, it is created for the user.

The default value is PRINTER.

USER_INFORMATION or *UI*

Specifies a user information string of up to 256 characters. This string enables you to pass information (such as a file path) to a backup job. This string is also passed on to all output files generated by the backup job.

If omitted, the user information string associated with the backup job is assumed.

Remarks

- If backup files are included in the log configuration, each repository switch for the log starts a job to back up the log. Each backup job uses the validation information specified on this subcommand.

- To determine if the ACCOUNT and PROJECT parameters are required and the valid JOB_CLASS values, display the validation information for the user name.

  To display validation information for a user name, use the Administer_User utility with the DISPLAY_USER subcommand. If you are logged in as the family administrator, you can display information on any user in the family; otherwise, you can display information only for the user name you are using.

  For more information about family administration and user validation see the NOS/VE User Validation manual and the NOS/VE System Usage manual.

- For more information see the NOS/VE Advanced File Management Usage manual.

SET_PERFORMANCE_OPTION ADMRL Subcommand

Purpose Specifies the performance emphasis (speed or reliability) for the log.

Format

SET_PERFORMANCE_OPTION or
SETPO
EMPHASIS=keyword
LOG_ENTRY=keyword
STATUS=status variable

Parameters

EMPHASIS or **E**

Specifies whether speed or reliability is more important.

SPEED or S

Speed is more important than reliability.

RELIABILITY or R

Reliability is more important than than speed.

BALANCED or B

Both speed and reliability are important.

This parameter is required.

LOG_ENTRY or *LOG_ENTRIES* or *LE*

Indicates the types of log entries to which the specified emphasis applies.

RECORD or R

Record entries, but not parcel entries.

PARCEL or P

For future implementation.

ALL or A

For future implementation.

The default value is RECORD.

Remarks

- This subcommand determines how frequently log entries in memory are written to disk. (Its purpose is similar to that of the FORCED_WRITE attribute for keyed files.)
- If this subcommand is not specified, the default performance option is BALANCED.
- The EMPHASIS values have the following meanings:

  SPEED

  The system memory manager determines when log entries are written to disk.

  RELIABILITY

  Each log entry is written to disk before the next log entry begins.

  BALANCED

  The system must begin writing a log entry to disk before the next log entry can begin.

- Any value specified for parcels is recorded for future use, but is currently ignored.
- For more information, see the NOS/VE Advanced File Management Usage manual.

Examples

The following session changes the performance options for $USER.MY_LOG.

```
/administer_recovery_log
admrl/use_log, $user.my_log
admrl/set_performance_option, ..
admrl../emphasis=reliability
admrl/quit
/
```

SET_VERIFICATION_LEVEL ADMRL Subcommand

Purpose Indicates whether checksums should be performed for the header and trailer parts of log records.

Format

SET_VERIFICATION_LEVEL or
SETVL
VERIFY_LOG_ENTRIES=boolean
STATUS=status variable

Parameters **VERIFY_LOG_ENTRIES** or **VLE**

Indicates whether checksums are performed for the log.

TRUE or YES or ON

Checksums are performed.

FALSE or NO or OFF

Checksums are not performed.

This parameter is required.

Remarks

- This subcommand can be specified only for a new log. The verification level cannot be changed for an existing log.
- This subcommand is optional. If it is omitted from a session that creates a new log, the default verification level is FALSE.
- For more information see the NOS/VE Advanced File Management Usage manual.

USE_LOG ADMRL Subcommand

Purpose Establishes the log to be created or changed by the session.

Format

USE_LOG or
USEL
CATALOG=file
STATUS=status variable

Parameters

CATALOG or C

Catalog path for the log created or changed by the session.

A session can create or change only one log; therefore, any subsequent USE_LOG subcommands are ignored.

If the catalog does not exist, the subcommand creates it. If the catalog exists, but does not contain a log, a log is created in it. If a log exists in the catalog, the session verifies that the log contains the proper characteristics.

This parameter is required.

Remarks

- You must establish a catalog before any of the other subcommands (except QUIT, DELETE_LOG, HELP, or CANCEL_LOG_CHANGES (after DELETE_LOG)) can be entered.
- Once established, the catalog can only be changed after using CANCEL_LOG_CHANGES.
- For more information see the NOS/VE Advanced File Management Usage manual.

Examples

The following session establishes $USER.MY_LOG as the log to be used. The performance options for $USER.MY_LOG are changed, but then the changes are cancelled and another log is specified.

```
/administer_recovery_log
admr1/use_log, $user.my_log
admr1/set_performance_option, emphasis=reliability
admr1/cancel_log_changes
admr1/use_log, $user.my_log_2
admr1/
```

ADMINISTER_VALIDATIONS 5

ADMINISTER_VALIDATIONS Command

Purpose Displays and changes validations.

Format **ADMINISTER_VALIDATIONS** or **ADMV**
STATUS=status variable

Remarks For more information, see the NOS/VE User Validation manual.

CHANGE_DEFAULT_ACCOUNT_PROJECT CREU and CHAU Subcommand

Purpose Changes the default account and project for the LOGIN and SUBMIT_JOB commands.

Format **CHANGE_DEFAULT_ACCOUNT_PROJECT** or **CHADAP**
ACCOUNT=name or *keyword*
PROJECT=name or *keyword*
STATUS=status variable

Parameters *ACCOUNT* or *A*

Specifies the new account name. The default is that the account is not changed. The keywords are:

DEFAULT

The account is set to the default value specified in the DEFAULT_ACCOUNT_PROJECT field description as defined by the family or system administrator.

CURRENT

The account of the job executing this command is used.

PROJECT or *P*

Specifies the new project name. The default is that the project is not changed. The keywords are:

DEFAULT

The project is set to the default value specified in the DEFAULT_ACCOUNT_PROJECT field description as defined by the family or system administrator.

CURRENT

The account of the job executing this command is used.

Remarks

- If the system is running at an account or project validation level, a warning error is returned if the specified account or project does not exist.
- For more information, see the NOS/VE User Validation manual.

Examples To change the default login account and project, enter:

```
ADMV/change_user
CHAU/change_default_account_project account=a project=b
CHAU/quit
```

CHANGE_LINK_ATTRIBUTE_CHARGE CREU and CHAU Subcommand

Purpose Changes the charge number needed to gain access to NOS or NOS/BE permanent files or to submit a job to NOS or NOS/BE.

Format

CHANGE_LINK_ATTRIBUTE_CHARGE or **CHALAC**
VALUE=string or *keyword*
STATUS=status variable

Parameters *VALUE* or *V*

Specifies the new NOS or NOS/BE charge number. The default is that the link attribute charge number is not changed. If you specify DEFAULT, the default value specified in the LINK_ATTRIBUTE_CHARGE field description as defined by the family or system administrator is used.

Remarks

- You can override this value by using the CHANGE_LINK_ATTRIBUTE command.
- You are responsible for maintaining the values in the LINK_ATTRIBUTE_CHARGE validation field.
- For more information, see the NOS/VE User Validation manual.

CHANGE_LINK_ATTRIBUTE_FAMILY CREU and CHAU Subcommand

Purpose Changes the family name needed to gain access to NOS or NOS/BE permanent files or to submit a job to NOS or NOS/BE.

Format **CHANGE_LINK_ATTRIBUTE_FAMILY** or **CHALAF**

VALUE=string or *keyword*
STATUS=status variable

Parameters *VALUE* or *V*

Specifies the new NOS or NOS/BE family name. The default is that the link attribute family is not changed. If you specify DEFAULT, the default value specified in the LINK_ATTRIBUTE_FAMILY field description as defined by the family or system administrator is used.

Remarks

- You can override this value by using the CHANGE_LINK_ATTRIBUTE command.
- You are responsible for maintaining the values in the LINK_ATTRIBUTE_FAMILY validation field.
- For more information, see the NOS/VE User Validation manual.

CHANGE_LINK_ATTRIBUTE_PASSWORD CREU and CHAU Subcommand

Purpose Changes the password needed to gain access to NOS or NOS/BE permanent files or to submit a job to NOS or NOS/BE.

Format **CHANGE_LINK_ATTRIBUTE_PASSWORD** or **CHALAPW**

VALUE=string or *keyword*
STATUS=status variable

Parameters *VALUE* or *V*

Specifies the new NOS or NOS/BE password. The default is that the link attribute password is not changed. If you specify DEFAULT, the default value specified in the LINK_ATTRIBUTE_PASSWORD field description as defined by the family or system administrator is used.

Remarks

- You can override this value by using the CHANGE_LINK_ATTRIBUTE command.
- You are responsible for maintaining the values in the LINK_ATTRIBUTE_PASSWORD validation field.
- For more information, see the NOS/VE User Validation manual.

CHANGE_LINK_ATTRIBUTE_PROJECT CREU and CHAU Subcommand

Purpose Changes the project number needed to gain access to NOS or NOS/BE permanent files or to submit a job to NOS or NOS/BE.

Format **CHANGE_LINK_ATTRIBUTE_PROJECT** or **CHALAP**

VALUE=string or *keyword*
STATUS=status variable

Parameters *VALUE* or *V*

Specifies the project number needed to gain access to NOS and NOS/BE permanent files or to submit a job to NOS or NOS/BE. The default is that the link attribute project is not changed. If you specify DEFAULT, the default value specified in the LINK_ATTRIBUTE_PROJECT field description as defined by the family or system administrator is used.

Remarks

- You can override this value by using the CHANGE_LINK_ATTRIBUTE command.
- You are responsible for maintaining the values in the LINK_ATTRIBUTE_PROJECT validation field.
- For more information, see the NOS/VE User Validation manual.

CHANGE_LINK_ATTRIBUTE_USER CREU and CHAU Subcommand

Purpose Changes the user name needed to gain access to NOS or NOS/BE permanent files or to submit a job to NOS or NOS/BE.

Format

CHANGE_LINK_ATTRIBUTE_USER or **CHALAU**
VALUE=string or *keyword*
STATUS=status variable

Parameters *VALUE* or *V*

Specifies the new NOS or NOS/BE user name. The default is that the link attribute user is not changed. If you specify DEFAULT, the default value specified in the LINK_ATTRIBUTE_USER field description as defined by the family or system administrator is used.

Remarks

- You can override this value by using the CHANGE_LINK_ATTRIBUTE command.
- You are responsible for maintaining the values in the LINK_ATTRIBUTE_USER validation field.
- For more information, see the NOS/VE User Validation manual.

CHANGE_LOGIN_PASSWORD CREU and CHAU Subcommand

Purpose Changes information about the user's login password.

Format

CHANGE_LOGIN_PASSWORD or
CHALPW
OLD_PASSWORD=name
NEW_PASSWORD=name
EXPIRATION_DATE=date_time or *keyword*
EXPIRATION_INTERVAL=integer or *keyword*
MAXIMUM_EXPIRATION_INTERVAL=integer or *keyword*
EXPIRATION_WARNING_INTERVAL=integer or *keyword*
ADD_ATTRIBUTES=list of name or *keyword*
DELETE_ATTRIBUTES=list of name or *keyword*
ENCRYPTED_PASSWORD=string
STATUS=status variable

Parameters

OLD_PASSWORD or *OPW*

Specifies the current login password. This parameter is required for a user if the NEW_PASSWORD parameter is specified. This parameter is optional for all administrators.

NEW_PASSWORD or *NPW*

Specifies the new login password for the user. The default is that the user's password is not changed.

EXPIRATION_DATE or *ED*

Specifies the date and time when this password expires. The number of days between the current date and what the EXPIRATION_DATE parameter specifies cannot exceed the number of days specified by the MAXIMUM_EXPIRATION_INTERVAL parameter.

The format is YYYY-MM-DD.HH:MM:SS. The hours, minutes, and seconds portion is optional, and the time defaults to midnight 00:00:00.

The default is that a new expiration date is calculated by adding the value specified by the EXPIRATION_INTERVAL parameter to the current date. You must use

a new password and this parameter to override the calculation of the EXPIRATION_DATE value. The keywords are:

NONE

The password for this user never expires.

DEFAULT

The expiration date is set to the default value specified in the LOGIN_PASSWORD field description as defined by the family or system administrator.

EXPIRATION_INTERVAL or *EI*

Specifies the number of days (1 to 365) until the password expires. When you change the password but don't specify the EXPIRATION_DATE parameter, the system calculates day the new password will expire by adding the value specified by the EXPIRATION_INTERVAL parameter to the current date and time. The number of days specified by the EXPIRATION_INTERVAL parameter must not exceed the MAXIMUM_EXPIRATION_INTERVAL parameter for this user. The default is that the current EXPIRATION_INTERVAL parameter is not changed. The keywords are:

UNLIMITED

The password will not expire until a specific date is specified by the EXPIRATION_DATE parameter.

DEFAULT

The expiration interval is set to the default value specified in the LOGIN_PASSWORD field description as defined by the family or system administrator.

MAXIMUM_EXPIRATION_INTERVAL or *MEI*

Specifies the an upper limit in days for the EXPIRATION_INTERVAL parameter. Only system administrators, family administrators, account members with user administration capability, or project members with user administration capability can specify this parameter.

EXPIRATION_WARNING_INTERVAL or *EWI*

Specifies the number of days (0 to 365) before the password expiration date that warnings are sent to the user user that the password will expire. If you specify zero, the user does not receive a warning. The default is that the current value is not changed. The keywords are:

UNLIMITED

The user always receives a warning during each login.

DEFAULT

The expiration warning interval is set to the default value specified in the LOGIN_PASSWORD field description as defined by the family or system administrator.

ADD_ATTRIBUTES or *AA*

Specifies a list of site-defined password attributes to be added. No attributes are released. Only system administrators, family administrators, account members with user administration capability, or project members with user administration capability can specify this parameter.

DELETE_ATTRIBUTES or *DA*

Specifies a list of site-defined password attributes to be deleted. No attributes are released. Only system administrators, family administrators, account members with user administration capability, or project members with user administration capability can specify this parameter.

ENCRYPTED_PASSWORD or *EPW*

Currently not supported. Passwords are encrypted, and it is impossible to enter an encrypted value for the password.

Remarks

- You can also change passwords using CHANGE_LOGIN_PASSWORD command.
- You cannot change your expiration date unless you also change your password.
- For more information, see the NOS/VE User Validation manual.

Examples

To change the password and set the expiration date, enter:

```
ADMV/change_user
CHAU/change_login_password
CHAU../old_password=example ..
CHAU../new_password=sample ..
CHAU../expiration_date=1987-12-10 ..
CHAU../expiration_interval=60 ..
CHAU/quit
ADMV/
```

This password expires in 60 days.

CHANGE_USER
ADMV Subcommand

Purpose Starts the CHANGE_USER subutility to change a user validation.

Format

CHANGE_USER or
CHAU
USER=name
STATUS=status variable

Parameters *USER* or *U*

Specifies the user name to change. The default is that the user name specified during login is used.

Remarks

- System and family administrators can change any user's validations. Account or project members with user administration capability can only change user validations for user names under their control. Users can change some of their own validations.
- For more information, see the NOS/VE User Validation manual.

Examples To change the default account and project for the LOGIN and SUBMIT_JOB commands, enter:

```
ADMV/change_user
CHAU/change_default_account_project account=a ..
CHAU../project=b
CHAU/quit
ADMV/
```

CHANGE_USER_EPILOG
CREU and CHAU Subcommand

Purpose Specifies the name of the user's epilog file.

Format

CHANGE_USER_EPILOG or
CHAUE
VALUE=any or *keyword*
STATUS=status variable

Parameters *VALUE* or *V*

Specifies the new file reference. You can specify a file reference value as a string or file path. A string or a file path produce equivalent results. The default is that the user epilog is not changed. The keywords are:

DEFAULT

The name of the user epilog is set to the default value specified in the USER_EPILOG field description as defined by the administrator.

NONE

The file reference $NULL is used. The user does not have an epilog file.

Remarks

- If you enter the file reference as a file path, it is translated to a full path before it is stored in the validation file. If you enter the file reference as a string, the string is stored in the validation file, and the full path name is completed when the epilog is called during job termination.

- For more information, see the NOS/VE User Validation manual.

Examples

To change your epilog that file ALL_DONE is used, enter:

```
ADMV/change_user
CHAU/change_user_epilog value=$user.all_done
CHAU/quit
ADMV/
```

CHANGE_USER_PROLOG CREU and CHAU Subcommand

Purpose Specifies the name of the user's prolog file.

Format

CHANGE_USER_PROLOG or
CHAUP
VALUE=any or *keyword*
STATUS=status variable

Parameters *VALUE* or *V*

Specifies the new file reference. You can specify a file reference value as a string or file path. A string or a file path produce equivalent results. The default is that the user prolog is not changed. The keywords are:

DEFAULT

The name of the user prolog is set to the default value specified in the USER_PROLOG field description as defined by the administrator.

NONE

The file reference $NULL is used. The user does not have a prolog file.

Remarks

- If you enter the file reference as a file path, it is translated to a full path before it is stored in the validation file. If you enter the file reference as a string, the string is stored in the validation file, and the full path name is completed when the epilog is called during job termination.

- For more information, see the NOS/VE User Validation manual.

Examples

To change your prolog so that file START_UP is used, enter:

```
ADMV/change_user
CHAU/change_user_prolog value=$user.start_up
CHAU/quit
ADMV/
```

DISPLAY_USER
ADMV Subcommand

Purpose Displays your validations.

Format

DISPLAY_USER or
DISPLAY_USERS or
DISU
USER=list of name or *keyword*
OUTPUT=file
DISPLAY_OPTION=list of name or *keyword*
STATUS=status variable

Parameters

USER or *USERS* or *U*

Lists the user names to display. The default is the user specified during login.

OUTPUT or *O*

Specifies the file to which the validations are written. The default is $OUTPUT.

DISPLAY_OPTION or *DISPLAY_OPTIONS* or *DO*

Lists the names of the user validations to display. The default is the keyword ALL. The keywords are:

ALL

The value of every user validation field is displayed.

NONE

Only the user names are displayed.

Remarks

- Each validation has an associated display authority that specifies who can display the value of the validation. If a user does not have enough authority to display the value, the message "Not authorized to display value" is written in place of the validation value.
- For more information, see the NOS/VE User Validation manual.

Examples To display all of the validations, enter:

```
ADMV/display_user
```

To display the default login account and project, enter:

```
ADMV/display_user all ..
ADMV../display_option=default_account_project
```

END_ADMINISTER_VALIDATIONS ADMV Subcommand

Purpose Terminates an ADMINISTER_VALIDATIONS utility session.

Format

END_ADMINISTER_VALIDATIONS or
ENDAV or
QUIT or
QUI
STATUS=status variable

Remarks For more information, see the NOS/VE User Validation manual.

END_CHANGE_USER CHAU Subcommand

Purpose Terminates a CHANGE_USER subutility session.

Format

END_CHANGE_USER or
ENDCU or
QUIT or
QUI
WRITE_CHANGES=boolean
STATUS=status variable

Parameters *WRITE_CHANGES* or *WC*

Specifies whether the changes made during the CHANGE_USER subutility session are written to the validation file. The default is TRUE. The keywords are:

TRUE

The changes are written to the validation file.

FALSE

No changes are written to the validation file.

Remarks For more information, see the NOS/VE User Validation manual.

ANALYZE_OBJECT_LIBRARY 6

ANALYZE_OBJECT_LIBRARY Command

Purpose Begins an ANALYZE_OBJECT_LIBRARY utility session. The subcommands for this object code utility display the internal characteristics of object modules, including: object record counts, section sizes, section attributes, and performance data for modules on an object library or object file.

Format

ANALYZE_OBJECT_LIBRARY or
ANAOL
LIBRARY=file
STATUS=status variable

Parameters *LIBRARY* or *L*

Object library or object file to be analyzed.

If LIBRARY is omitted, you must use the USE_LIBRARY subcommand to specify the object library or object file.

Remarks

- After entering the ANALYZE_OBJECT_LIBRARY command, you can enter any of the ANAOL subcommands. The ANAOL session ends when you enter the QUIT subcommand.
- An object library or file must be specified on the ANALYZE_OBJECT_LIBRARY command or on the USE_LIBRARY subcommand before an ANAOL session can continue.
- For more information, see the NOS/VE Object Code Management manual.

Examples The following is a sequence that enters the ANALYZE_OBJECT_LIBRARY utility, specifies LGO as the file to be analyzed, and displays the characteristics of library LGO.

```
/analyze_object_library lgo
AOL/display_library_analysis
Library Analysis of LGO
Number of modules:  2
Record Analysis

    Identification records:   2
    Libraries:                2 - items:    10
    Section definitions:      9
    Text records:             21 - items:   519
       .
       .
    Relocation records:       2 - items:    8
    Binding templates:        8
    Transfer symbols:         2
                              ---
      Total records:          84

AOL/quit
```

DISPLAY_LIBRARY_ANALYSIS ANAOL Subcommand

Purpose Displays the number of modules and/or the total number of each type of object record on the current object library or file. The current object library or file is specified by a previous USE_LIBRARY subcommand or ANALYZE_OBJECT_LIBRARY command.

Format

DISPLAY_LIBRARY_ANALYSIS or
DISLA
DISPLAY_OPTIONS=list of keyword
OUTPUT=file
STATUS=status variable

Parameters *DISPLAY_OPTIONS* or *DISPLAY_OPTION* or *DO*

List of one or more keywords indicating the analysis information to be displayed. Options are:

NUMBER_OF_MODULES or NOM

Number of modules on the object library or file.

RECORD_ANALYSIS or RA

Total number of each type of object record on the object library or file.

ALL

All of the previously listed options.

IF DISPLAY_OPTION is omitted, all analysis information is displayed.

OUTPUT or *O*

Output file. This file can be positioned.

If OUTPUT is omitted, file $OUTPUT is used.

Remarks

- In a library analysis (see example), the record analysis contains the number of each type of object record in the library or file. The total number of adaptable items is also listed with the object records that have adaptable fields.
- For more information, see the NOS/VE Object Code Management manual.

Examples

The following ANAOL session lists the number of modules and the type and number of object records in the current library LGO.

```
/analyze_object_library lgo
AOL/display_library_analysis

Library Analysis of LGO

  Number of modules:  2

  Record Analysis

    Identification records:              2
    Libraries:                           2   items:      10
    Section definitions:                 9
    Text records:                       21   items:     519
    Address formulation records:        31   items:      31
    External linkage records:            5   items:       5
    Entry definitions:                   2
    Relocation records:                  2   items:       8
    Binding templates:                   8
    Transfer symbols:                    2
                                    --------
      Total records:                    84

AOL/
```

DISPLAY_MODULE_ANALYSIS ANAOL Subcommand

Purpose Displays analysis information about specified modules on the object library or file, such as:

- Total number of each type of object record in the module.
- Size, type, attributes initialized, addresses in, externals in, and addresses to each section in the module.

The current object library or file is specified by a previous USE_LIBRARY subcommand or ANALYZE_OBJECT_LIBRARY command.

Format

DISPLAY_MODULE_ANALYSIS or
DISMA
MODULES=list of range of any
DISPLAY_OPTIONS=list of keyword
OUTPUT=file
STATUS=status variable

Parameters *MODULES* or *MODULE* or *M*

List of modules whose analysis information is to be displayed.

You use a string value for a module whose name is not an SCL name.

If MODULE is omitted or the keyword ALL is used, analysis information for all modules in the object library or file is displayed.

DISPLAY_OPTIONS or *DISPLAY_OPTION* or *DO*

List of one or more keywords indicating the analysis information to be displayed. Options are:

RECORD_ANALYSIS or RA

Total number of each type of object record in the module.

SECTION_ANALYSIS or SA

Size, type, attributes, bytes initialized, addresses built in this section, and addresses built in other sections that the loader will build that point to this section.

ALL

All of the previously listed options.

If DISPLAY_OPTION is omitted, all analysis information is displayed.

OUTPUT or *O*

Output file. This file can be positioned. If OUTPUT is omitted, file $OUTPUT is used.

Remarks

- In a module analysis display, the record analysis contains the number of each type of object record in the module. The total number of adaptable items is also listed with the object records that have adaptable fields. The number of items contained in the next column lists the total size of the adaptable record types.
- The section analysis display includes the following:
  - Total number of bytes in the section.
  - Section type: code section, binding section, working storage section, common block, extensible working storage, and extensible common block.
  - Attributes of the section: R=read, W=write, X=execute, and B=binding.
  - Number of bytes initialized in the section by text and replication records or by allotted text.
  - Number of internal addresses (Addresses in) the loader will build in this section.
  - Number of addresses (Addresses to) in other sections the loader will build that point to this section.
- For more information, see the NOS/VE Object Code Management manual.

Examples The following subcommand lists the record analysis and section analysis of module TEST.

```
AOL/display_module_analysis module=test

Module Analysis of TEST

  Record Analysis

    Identification records:              1
    Libraries:                           1   items:          5
    Section definitions:                 4
    Text records:                        9   items:        233
    Address formulation records:        15   items:         15
    External linkage records:            2   items:          2
    Entry definitions:                   1
    Relocation records:                  1   items:          4
    Binding templates:                   4
    Transfer symbols:                    1
                                    --------
      Total records:                    39

  Section Analysis

    Section: TEST      60 bytes       CODE [ R X ]
      Bytes initialized: 60 Addresses to: 1
    Section:           56 bytes       BINDING [ B ]
      Externals in: 2 Addresses in: 3 Addresses to: 1
    Section:           207 bytes       WORKING STORAGE [ R ]
      Bytes initialized: 163 Addresses in: 6 Addresses to: 8
    Section:           104 bytes       WORKING STORAGE [ R W ]
      Bytes initialized: 10 Addresses in: 7 Addresses to: 5

AOL/
```

DISPLAY_PERFORMANCE_DATA ANAOL Subcommand

Purpose Displays possible load and execution time performance problems that may exist in specified modules on the object library or file. The current object library or file is specified by a previous USE_LIBRARY subcommand or ANALYZE_OBJECT_LIBRARY command.

Format **DISPLAY_PERFORMANCE_DATA** or **DISPD**

MODULES=list of range of any
PERFORMANCE_DATA=list of keyword
DISPLAY_OPTION=list of keyword
OUTPUT=file
STATUS=status variable

Parameters

MODULES or *MODULE* or *M*

List of modules whose performance data is to be displayed.

You use a string value for a module whose name is not an SCL name.

If MODULE is omitted or keyword ALL is specified, performance data for all modules is displayed.

PERFORMANCE_DATA or *PD*

List of one or more keywords indicating the performance data to be displayed. Options are:

SYMBOL_TABLES or ST

Modules that have debug symbol tables.

LINE_TABLES or LT

Modules that have debug line address tables.

PARAMETER_CHECKING or PC

Modules that have parameter checking records.

RUNTIME_CHECKING or RC

Modules that have run-time range checking for variables, subscripts, and substring character expressions.

RUNTIME_LIBRARY_CALLS or RLC

Modules that have calls to local run-time libraries.

RUNTIME_LIBRARIES or RL

Modules that have text-embedded run-time library directives.

OPT_DEBUG or OD

Modules that are compiled with the parameter OPTIMIZATION_LEVEL=DEBUG.

OPT_LOW or OL

Modules that are compiled with the parameter OPTIMIZATION_LEVEL=LOW.

OBJECT_MODULES or OM

Object modules that are not on an object library.

LOAD_MODULES or LM

Load modules that have not been bound.

BOUND_MODULES or BM

Bound modules that have not been prelinked.

UNREFERENCED_SECTIONS or US

Modules that have uninitialized and unreferenced sections.

MULTIPLE_ENTRY_POINTS or MEP

Bound or prelinked modules that have multiple entry points.

ALL

All of the previously listed options.

If PERFORMANCE_DATA is omitted, all performance data is displayed.

DISPLAY_OPTION or *DISPLAY_OPTIONS* or *DO*

List of one or more keywords indicating the information to be displayed. The number of modules with the possible performance problem is always displayed. Options are:

NONE

No information other than the number of modules with the possible performance problem.

MODULE_NAMES or MN

Names of modules with the possible performance problem.

DESCRIPTION or D

Brief description of the possible performance problem and recommended changes to correct the problem.

ALL

All of the previously listed options.

If DISPLAY_OPTION is omitted, the number of modules with the possible problem and the description of the problem (DESCRIPTION) are displayed.

OUTPUT or *O*

Output file. This file can be positioned.

If OUTPUT is omitted, file $OUTPUT is used.

Remarks

- The analysis performed is very general, and the recommendations may not be applicable to all programs. Each recommendation should be looked at to determine if any changes should be made to the program or its packaging.
- The quality of analysis performed depends on the amount of information placed in the object modules by the compilers. Some modules may have performance problems that are not detected.
- Since binding and prelinking may hide some of a product's performance problems, analysis should also be done on the unbound product.
- For more information, see the NOS/VE Object Code Management manual.

DISPLAY_SECTION_ANALYSIS ANAOL Subcommand

Purpose

Displays section usage information for specified modules on the object library or file. Information displayed includes size, attributes, bytes initialized, addresses in the section, and addresses to the section. The current object library or file is specified by a previous USE_LIBRARY subcommand or ANALYZE_OBJECT_LIBRARY command.

Format

DISPLAY_SECTION_ANALYSIS or
DISSA
MODULES=list of range of any
SECTION_KINDS=list of keyword
SECTION_ACCESS_ATTRIBUTES=list of keyword
SECTION_NAME=name
OUTPUT=file
STATUS=status variable

Parameters *MODULES* or *MODULE* or *M*

List of modules whose section usage information is to be displayed.

Use a string value for a module whose name is not an SCL name.

If MODULE is omitted or the keyword ALL is used, section usage information for all modules in the object library or file is displayed.

SECTION_KINDS or *SK*

List of one or more keywords indicating the type of section to be displayed. Types are:

CODE or C

Code section.

BINDING or B

Binding section.

WORKING_STORAGE or WS

Working storage section.

EXTENSIBLE_WORKING_STORAGE or EWS

Extensible working storage section.

COMMON_BLOCK or CB

Common block section.

EXTENSIBLE_COMMON_BLOCK or ECB

Extensible common block section.

ALL

All of the previously listed section types:

If SECTION_KIND is omitted, all section types are displayed.

SECTION_ACCESS_ATTRIBUTES or *SAA*

List of one or more keywords indicating the access attributes of the section to be displayed. The access attributes are:

READ or R

Read attributes.

WRITE or W

Write attributes.

EXECUTE or E

Execute attributes.

BINDING or B

Binding attributes.

ALL

Any of the listed attributes.

If SECTION_ACCESS_ATTRIBUTE is omitted, sections with any attributes are displayed.

SECTION_NAME or *SN*

The name of the section to be displayed. If SECTION_NAME is omitted, sections with any names are displayed.

Use a string value for a section whose name is not an SCL name.

OUTPUT or *O*

Output file. This file can be positioned.

If OUTPUT is omitted, file $OUTPUT is used.

Remarks

The section analysis display (see example) includes the following:

- Name of section (if any).
- Total number of bytes in the section.
- Section type: code section, binding section, working storage section, common block, extensible working storage, and extensible common block.
- Attributes of the section: R=read, W=write, X=execute, B=binding.
- Number of bytes initialized in the section by text and replication records or by allotted text.

- Number of internal addresses (Addresses in) and external addresses (Externals in) the loader will build in this section.
- Number of addresses (Addresses to) in other sections which the loader will build that point to this section.
- For more information, see the NOS/VE Object Code Management manual.

Examples The following subcommand lists the section definitions for module SUB.

```
AOL/display_section_analysis module=sub

Section Usage of SUB
  Section: SUB       50 bytes            CODE [ R X ]
    Bytes initialized: 50
  Section:           24 bytes            BINDING [ B ]
    Externals in: 1 Addresses in: 1
  Section:           125 bytes            WORKING STORAGE [ R ]
    Bytes initialized: 101 Addresses in: 4 Addresses to: 2
  Section:           64 bytes            WORKING STORAGE [ R W ]
    Bytes initialized: 12 Addresses in: 2 Addresses to: 5

AOL/
```

QUIT
ANAOL Subcommand

Purpose Ends the ANALYZE_OBJECT_LIBRARY session.

Format **QUIT** or
QUI

Parameters None.

Remarks For more information, see the NOS/VE Object Code Management manual.

Examples The following sequence writes a library and a module analysis of LIBRARY_1 to file OUT1 and writes a library analysis of OBJECT_FILE_2 to file OUT2. The output files are then printed.

```
/analyze_object_library library_1
AOL/display_library_analysis output=out1
AOL/display_module_analysis display_option=..
AOL../section_analysis output=out1.$eoi
AOL/use_library object_file_2
AOL/display_library_analysis output=out2
AOL/quit
/print_file out1
/print_file out2
```

USE_LIBRARY
ANAOL Subcommand

Purpose Specifies the object library or object file to be analyzed.

Format

USE_LIBRARY or
USEL
LIBRARY=file
STATUS=status variable

Parameters

LIBRARY or **L**

Object library or object file to be analyzed. This parameter is required.

Remarks

- If an object library or object file was not specified on the ANALYZE_OBJECT_LIBRARY command, you must specify the library or file with the USE_LIBRARY subcommand before you can analyze the library, its modules, or its sections.
- You use this subcommand to specify a new object library or object file to analyze.
- For more information, see the NOS/VE Object Code Management manual.

Examples The following subcommand selects object file LGO as the next library to be analyzed.

```
AOL/use_library lgo
```

BACKUP_PERMANENT_FILES 7

BACKUP_PERMANENT_FILES Command

Purpose Initiates execution of the utility that backs up permanent files and catalogs. Further processing is directed by utility subcommands.

Format

BACKUP_PERMANENT_FILES or
BACKUP_PERMANENT_FILE or
BACPF
BACKUP_FILE=file
LIST=file
STATUS=status variable

Parameters

BACKUP_FILE or **BF**

Specifies the file to which backup information is copied. You can specify a file position of beginning-of-information or end-of-information if the file is a mass storage file or a labelled tape. If no file position is specified, or the file is an unlabelled tape, the file is initially positioned to beginning-of-information. This parameter is required.

LIST or *L*

Identifies the file to which a summary of the results of executing the backup utility is written and, optionally, specifies how the file is to be positioned prior to use. Omission causes $LIST to be used.

Remarks

- You can back up only the files for which you have read access.
- For more information, see the NOS/VE System Usage manual.

Examples The following command initiates a BACKUP_PERMANENT_FILE command utility session. The command specifies that the backed up files are to be written to file BACKED_UP_FILES with the report listing written to file BACKUP_LISTING.

```
/backup_permanent_files bf=backed_up_files ..
../l=backup_listing
```

Following the entry of this command, BACKUP_PERMANENT_FILE subcommands can be entered in response to the following prompt.

```
PUB/
```

BACKUP_CATALOG BACPF Subcommand

Purpose Creates a backup copy of each file cycle and catalog registered in a specified catalog.

Format

BACKUP_CATALOG or
BACC
CATALOG=file
STATUS=status variable

Parameters

CATALOG or **C**

Specifies the catalog to be backed up. This parameter is required.

Remarks

- Starting at the specified catalog, the complete catalog hierarchy is followed to obtain a backup copy of each file and its associated catalog information.
- You must have READ access to the files in the catalog to be backed up, and not be required to share the files for APPEND, MODIFY or SHORTEN access.
- If you are not the owner of the catalog, backup copies for all file cycles (and their associated catalogs) to which you have read access and only for those files that have null passwords are made.
- BACKUP_CATALOG skips a file cycle if the file cycle is busy (that is, if it cannot access the file with an access mode of read and a share mode of read and execute).
- Previous EXCLUDE_CATALOG and EXCLUDE_FILE subcommands enable you to exclude catalogs and files from the backup operation.

- Previous INCLUDE_CYCLES, INCLUDE_VOLUME, INCLUDE_LARGE_CYCLES, and EXCLUDE_HIGHEST_CYCLE subcommands can limit the number of cycles actually backed up with the BACKUP_CATALOG subcommand.

- For more information, see the NOS/VE System Usage manual.

Examples The following command and subcommands back up all files in the master catalog.

```
/backup_permanent_files bf=back_up_files ..
PUB../list=backup_listing
PUB/backup_catalog c=$user
PUB/quit
```

BACKUP_FILE
BACPF Subcommand

Purpose Creates a backup copy of a specified permanent file.

Format

BACKUP_FILE or
BACF
FILE=file
PASSWORD=name or *keyword*
STATUS=status variable

Parameters

FILE or **F**

Specifies the permanent file or permanent file cycle for which a backup copy is to be made. This parameter is required.

PASSWORD or *PW*

Specifies the password of the file to be backed up. If you omit this parameter, or specify the keyword NONE, no password is used.

Remarks

- If the FILE parameter specifies a cycle reference, only that cycle is backed up. If a cycle reference is omitted, all cycles of the file are backed up.

- You must have READ access to the files to be backed up and not be required to share the files for APPEND, MODIFY, or SHORTEN access.

- BACKUP_FILE skips a file cycle if the file cycle is busy (that is, if it cannot access the file with an access mode of read and a share mode of read and execute).
- A previous EXCLUDE_FILE subcommand can be used to exclude specific cycles from the backup operation.
- Previous INCLUDE_CYCLES, INCLUDE_VOLUME, INCLUDE_LARGE_CYCLES, and EXCLUDE_HIGHEST_CYCLE subcommands can limit the number of cycles actually backed up with the BACKUP_FILE subcommand.
- For more information, see the NOS/VE System Usage manual.

Examples The following example backs up cycle number 87 of file DATA_FILE_0 in subcatalog CATALOG_1 of the master catalog.

```
/bacpf bf=copy_of_file
PUB/backup_file $user.catalog_1.data_file_0.87 ..
PUB../pw=new_data_0_pw
PUB/quit
```

DELETE_CATALOG_CONTENTS BACPF Subcommand

Purpose Deletes all files and subcatalogs in a catalog.

Format

DELETE_CATALOG_CONTENTS or
DELETE_CATALOG_CONTENT or
DELCC
CATALOG=**file**
STATUS=*status variable*

Parameters

CATALOG or **C**

Specifies the catalog whose contents is to be deleted. This parameter is required.

Remarks

- Only the owner of a catalog can use this subcommand to delete a catalog and to delete files with nonnull passwords.
- Alternate users can use this request to delete all files:
  - to which they have control and read access permission.
  - that they are not required to share for modify, shorten, and append access.
  - that have null passwords.
- If a file cycle is in use at the time this subcommand is entered, the actual delete is not done until the last user detaches the file.
- Previous EXCLUDE_CATALOG, EXCLUDE_FILE, EXCLUDE_HIGHEST_CYCLES, INCLUDE_CYCLES, INCLUDE_LARGE_CYCLES, INCLUDE_VOLUME, and INCLUDE_EMPTY_CATALOGS subcommands can be used to specify a subset of the permanent files to be deleted.
- DELETE_CATALOG_CONTENT skips a file cycle if the file cycle is busy (that is, if it cannot access the file with an access mode of read and a share mode of read and execute).
- For more information, see the NOS/VE System Usage manual.

Examples

The following example deletes the contents of catalog CATALOG_1 for the current user.

```
/backup_permanent_files bf=backup_of_files
PUB/delcc $user.catalog_1
```

DELETE_FILE_CONTENTS BACPF Subcommand

Purpose Deletes all cycles of a file.

Format

DELETE_FILE_CONTENTS or
DELETE_FILE_CONTENT or
DELFC
FILE=**file**
PASSWORD=*name* or *keyword*
STATUS=*status variable*

Parameters

FILE or **F**

Specifies the file to be deleted. The cycle number is ignored. This parameter is required.

PASSWORD or *PW*

Specifies the file password of the file to be deleted. This name must match the password registered with the file. Omission or specifying the keyword NONE causes no password to be used.

Remarks

- Only the owner of the file or a user with control and read access permission and a share mode permission that does not include modify, shorten, or append can delete a file.
- DELETE_FILE_CONTENT skips a file cycle if the file cycle is busy (that is, if it cannot access the file with an access mode of read and a share mode of read and execute).
- If a file cycle is in use at the time this subcommand is entered, the actual delete is not done until the last user detaches the file.
- Previous EXCLUDE_FILE, EXCLUDE_HIGHEST_CYCLES, INCLUDE_VOLUME, INCLUDE_LARGE_CYCLES, and INCLUDE_CYCLES subcommands can be used to specify a subset of the permanent file cycles to be deleted.
- For more information, see the NOS/VE System Usage manual.

Examples The following example deletes all cycles of permanent file DATA_FILE_1 for the current user.

```
/bacpf backup_of_files
PUB/delete_file_contents $user.data_file_1
```

EXCLUDE_CATALOG BACPF Subcommand

Purpose Excludes a catalog from subsequent backup and delete operations.

Format

EXCLUDE_CATALOG or
EXCC
CATALOG=file
STATUS=status variable

Parameters

CATALOG or C

Specifies the catalog that is to be excluded from subsequent backup and delete operations. This parameter is required.

Remarks

- This subcommand takes precedence over all INCLUDE subcommands.
- The catalog is excluded only if the subsequent backup operation is at a higher level in the catalog hierarchy; thus, you can override this subcommand by explicitly backing up a catalog that is at a lower level in the catalog hierarchy.
- For more information, see the NOS/VE System Usage manual.

EXCLUDE_FILE BACPF Subcommand

Purpose Excludes a file or cycle from subsequent backup and delete operations.

Format

EXCLUDE_FILE or
EXCF
FILE=file
STATUS=status variable

Parameters FILE or F

Specifies the file or cycle that is to be excluded from subsequent backup and delete operations. This parameter is required.

Remarks

- This subcommand takes precedence over all INCLUDE subcommands.
- The file or cycle is excluded only if the subsequent backup or delete operation is at a higher level in the catalog hierarchy; thus, you can override this subcommand by explicitly backing up the file or cycle.
- For more information, see the NOS/VE System Usage manual.

EXCLUDE_HIGHEST_CYCLES BACPF Subcommand

Purpose Causes the specified number of high (largest numbered) cycles of permanent files to be excluded from subsequent backup and delete operations.

Format

EXCLUDE_HIGHEST_CYCLES or
EXCLUDE_HIGHEST_CYCLE or
EXCHC
NUMBER_OF_CYCLES=integer or *keyword*
STATUS=status variable

Parameters *NUMBER_OF_CYCLES* or *NOC*

Specifies the number of high cycles to be excluded. The value must be an integer in the range from 0 through 999. Omission causes 3 to be used.

Remarks

- This subcommand takes precedence over all INCLUDE subcommands.
- For more information, see the NOS/VE System Usage manual.

Examples The following example excludes the highest cycle of each file in a user's catalog from a subsequent DELETE_CATALOG_CONTENTS command.

```
/bacpf bf=backup_of_files
PUB/exclude_highest_cycles noc=1
PUB/delete_catalog_contents $user
```

INCLUDE_CYCLES
BACPF Subcommand

Purpose Includes cycles in subsequent backup and delete operations based on the creation date and time, last access date and time, last modification date and time, or expiration date of the cycle.

Format

INCLUDE_CYCLES or
INCLUDE_CYCLE or
INCC
SELECTION_CRITERIA=keyword
MONTH=integer or *keyword*
DAY=integer
YEAR=integer
HOUR=integer
MINUTE=integer
SECOND=integer
MILLISECOND=integer
STATUS=status variable

Parameters

SELECTION_CRITERIA or **SC**

Specifies the selection criteria to be used in determining which cycles will be backed up on subsequent backup and delete operations. This parameter is required. The following keywords can be specified:

ACCESSED_BEFORE

Only cycles whose last access was before the given date and time are selected.

ACCESSED_AFTER

Only cycles whose last access was after the given date and time are selected.

CREATED_BEFORE

Only cycles created before the given date and time are selected.

CREATED_AFTER

Only cycles created after the given date and time are selected.

EXPIRED_BEFORE

Only cycles whose expiration date is before the given date are selected. If no date is specified, the current date is used.

EXPIRED_AFTER

Only cycles whose expiration date is after the given date are selected.

MODIFIED_BEFORE

Only cycles that were last modified prior to the given date and time are selected.

MODIFIED_AFTER

Only cycles that were last modified after the given date and time are selected.

ALL

Removes the effect of any previous INCLUDE_CYCLES subcommand. No date or time parameters can be specified with this selection.

MONTH or *M*

Specifies the month. This parameter must be given as an integer from 1 through 12 (corresponding to the months January through December) or as one of the following keyword values.

JANUARY
FEBRUARY
MARCH
APRIL
MAY
JUNE
JULY
AUGUST
SEPTEMBER

OCTOBER
NOVEMBER
DECEMBER

This parameter is required unless a SELECTION_CRITERIA of EXPIRED_BEFORE is specified, in which case omission causes the current month to be used.

If a SELECTION_CRITERIA of ALL is specified, this parameter must not be specified.

DAY or *D*

Specifies the day of the month. This parameter must be given as an integer from 1 through 31.

This parameter is required unless a SELECTION_CRITERIA of EXPIRED_BEFORE is specified, in which case omission causes the current day to be used.

If a SELECTION_CRITERIA of ALL is specified, this parameter must not be specified.

YEAR or *Y*

Specifies the year. This parameter must be an integer in the range from 1983 through 1999.

This parameter is required unless a SELECTION_CRITERIA of EXPIRED_BEFORE is specified, in which case omission causes the current year to be used.

If a SELECTION_CRITERIA of ALL is specified, this parameter must not be specified.

HOUR or *HR*

Specifies the hour of the day. This parameter must be an integer in the range from 0 through 23 (where 0 is midnight and 23 is 11 p.m.). Omission causes 0 to be used.

MINUTE or *MIN*

Specifies the minute of the hour. This parameter must be an integer in the range from 0 through 59. Omission causes 0 to be used.

SECOND or *SEC*

Specifies the second of the minute. This parameter must be an integer in the range from 0 through 59. Omission causes 0 to be used.

MILLISECOND or *MSEC*

Specifies the millisecond part of the second. This parameter must be an integer in the range from 0 through 999. Omission causes 0 to be used.

Remarks

- If a before criteria is selected, only cycles created, last accessed, last modified, or expired prior to the given date and time are backed up or deleted; if an after criteria is selected, only cycles created, last accessed, last modified, or expired after the given date and time are backed up or deleted.
- The following values for the selection criteria can be used together in this manner:

  ACCESSED_AFTER and ACCESSED_BEFORE
  CREATED_AFTER and CREATED_BEFORE
  EXPIRED_AFTER and EXPIRED_BEFORE
  MODIFIED_AFTER and MODIFIED_BEFORE

- INCLUDE_CYCLES provides the capability to perform a partial permanent file backup.
- To reduce the size of the permanent file base by deleting permanent file cycles that have not been accessed since the specified date and time, an INCLUDE_CYCLES subcommand with an accessed before selection criteria followed by a BACKUP_ALL_FILES and a DELETE_ALL_FILES subcommand can be used.
- Any EXCLUDE subcommands take precedence over this subcommand.
- For more information, see the NOS/VE System Usage manual.

Examples

The following example produces a partial backup composed of all files that were modified on or after January 2, 1987.

```
/backup_permanent_files bf=backup_of_files
PUB/include_cycles modified_after 1 2 1987
```

Two INCLUDE_CYCLES subcommands can be used to specify a window for backup and delete operations. For example:

```
PUB/include_cycles modified_after january 1 1987
PUB/include_cycles modified_before january 3 1987
```

These subcommands cause only those cycles modified on January 1, 1987, and January 2, 1987 to be selected for backup and delete operations.

The following example backs up and deletes all files that have not been accessed since May 1, 1986.

```
PUB/include_cycles accessed_before ..
.. /may 1, 1986
PUB/backup_catalog c=$user
PUB/delete_all_files
```

INCLUDE_EMPTY_CATALOGS BACPF Subcommand

Purpose Specifies whether or not subsequent DELETE_CATALOG_CONTENTS subcommands should delete empty catalogs.

Format

INCLUDE_EMPTY_CATALOGS or
INCLUDE_EMPTY_CATALOG or
INCEC
DELETE_CATALOGS=boolean
STATUS=status variable

Parameters *DELETE_CATALOGS* or *DELETE_CATALOG* or *DC*

Specifies whether or not empty catalogs encountered during a subsequent DELETE_ALL_FILES or DELETE_CATALOG_CONTENTS subcommand should be deleted. Omission causes TRUE to used.

Remarks

- This subcommand must be entered during a BACKUP_PERMANENT_FILES command utility session.
- If this subcommand is not issued prior to a DELETE_ALL_FILES or DELETE_CATALOG_CONTENTS subcommand, empty catalogs are not deleted when those subcommands are entered.
- For more information, see the NOS/VE System Usage manual.

Examples The following example deletes all catalogs in subcatalog CATALOG_1 of a user's master catalog.

```
PUB/include_empty_catalogs
PUB/delete_catalog_contents ..
PUB../$user.catalog_1
```

The following example saves empty catalogs from being deleted for user DLH in family FAMILY1.

```
PUB/include_empty_catalogs dc=false
PUB/delete_catalog_contents :family1.dlh
```

INCLUDE_LARGE_CYCLES BACPF Subcommand

Purpose Specifies that subsequent backup and delete operations should include only permanent file cycles whose size is greater than or equal to a specified number of bytes. An excluded cycle is not backed up or deleted, regardless of its size.

Format

INCLUDE_LARGE_CYCLES or
INCLUDE_LARGE_CYCLE or
INCLC
MINIMUM_SIZE=integer
STATUS=status variable

Parameters **MINIMUM_SIZE** or **MS**

Specifies the minimum size in bytes of cycles included on subsequent backup and delete operations. This parameter is required.

Remarks For more information, see the NOS/VE System Usage manual.

Examples The following example backs up and deletes all cycles greater than or equal to 1,000,000 bytes in size.

```
PUB/include_large_cycles ms=1000000
PUB/backup_catalog c=$use r
PUB/delete_all_files
```

INCLUDE_SMALL_CYCLES BACPF Subcommand

Purpose Specifies that subsequent backup and delete operations should include only permanent file cycles whose size is less than or equal to a specified number of bytes. An excluded cycle is not backed up or deleted, regardless of its size.

Format

INCLUDE_SMALL_CYCLES or
INCLUDE_SMALL_CYCLE or
INCSC
MAXIMUM_SIZE=integer or **keyword**
STATUS=status variable

Parameters

MAXIMUM_SIZE or **MS**

Specifies the maximum size in bytes of cycles included on subsequent backup and delete operations. The keyword MAXIMUM specifies that no limit is placed on the size of cycles included in subsequent backup commands. This parameter is required.

Remarks For more information, see the NOS/VE System Usage manual.

Examples The following example backs up and deletes all cycles less than or equal to 1,000,000 bytes in size.

```
PUB/include_small_cycles ms=1000000
PUB/backup_catalog c=$use r
PUB/delete_all_files
```

INCLUDE_VOLUMES BACPF Subcommand

Purpose Specifies which permanent file cycles included in a specified volume are to be backed up or deleted by subsequent backup operations.

Format

INCLUDE_VOLUMES or
INCLUDE_VOLUME or
INCV
RECORDED_VSNS=list of name or **keyword**
CYCLE_SELECTION=keyword
STATUS=status variable

Parameters

RECORDED_VSNS or RECORDED_VSN or RVSN

Specifies the volumes to include; must be a name of from 1 to 6 characters or the keyword ALL. The RECORDED_VSN specified when the volume was initialized must be supplied. This parameter is required.

CYCLE_SELECTION or CS

Specifies which cycles on a volume should be backed up. Options are:

INITIAL_VOLUME (IV)

Backup only the cycles whose beginning of information (BOI) is on the volume. Cycles whose BOI is on another volume are skipped.

MULTIPLE_VOLUMES (MV)

Backup all cycles which reside either partially or completely on the volume.

If CYCLE_SELECTION is omitted, MULTIPLE_VOLUMES is used.

Remarks

- The CYCLE_SELECTION parameter is ignored when the keyword ALL is specified on the RECORDED_VSN parameter.
- If you select the MULTIPLE_VOLUMES option and cycles reside on more than one volume and each volume is backed up by a different backup, then cycles will be redundantly backed up. If the system fails due to a permanent file device failure, you may reload the lost cycles with the Permanent File Restore utility's RESTORE_EXCLUDED_FILE_CYCLES subcommand on just the backup tapes containing the cycles of the failed device.
- If you select the INITIAL_VOLUME option, data will not be redundantly backed up. Hence, all volumes in a backup must be read when a restore operation is done after a device failure.
- For more information, see the NOS/VE System Usage manual.

Examples The following example backs up all files that reside on the disk volume VOL033, and then deletes and restores the files so that they are dispersed over all volumes in the permanent file system.

```
/backup_permanent_files bf=temp_backup
PUB/include_volume rvsn=VOL033 cs=mv
PUB/backup_catalog $user
PUB/delete_catalog_contents $user
PUB/quit
/restore_permanent_files
PUR/restore_existing_catalog ..
PUR../$user bf=temp_backup
PUR/quit
/
```

QUIT
BACPF Subcommand

Purpose Ends a BACKUP_PERMANENT_FILES utility session.

Format
QUIT or
QUI

Parameters None.

Remarks For more information, see the NOS/VE System Usage manual.

SET_BACKUP_OPTIONS
BACPF Subcommand

Purpose Specifies actions to be taken by the BACKUP_PERMANENT_FILE utility.

Format
SET_BACKUP_OPTIONS or
SET_BACKUP_OPTION or
SETBO
EXCLUDE_CATALOG_INFORMATION=boolean or *keyword*
NULL_BACKUP_FILE_OPTION=keyword
INCLUDE_ARCHIVE_INFORMATION=boolean or *keyword*
INCLUDE_DATA=list of keyword
STATUS=status variable

Parameters

EXCLUDE_CATALOG_INFORMATION or *ECI*

Reserved for the site administrator's use. See the NOS/VE System Performance and Maintenance manual, volume 2 for more information.

NULL_BACKUP_FILE_OPTION or *NBFO*

Specifies whether the BACKUP_PERMANENT_FILE utility should attempt to read file data during backups to $NULL or any file assigned to the NULL device class. If $NULL was not specified on the BACKUP_FILE parameter on the BACKUP_PERMANENT_FILES command, then this parameter has no effect. The keywords are:

READ_DATA (RD)

Specifies that BACKUP_PERMANENT_FILES should attempt to read all file data when backing up to $NULL.

UNSPECIFIED

Specifies that BACKUP_PERMANENT_FILES should not attempt to read all file data when backing up to $NULL. For this option, the only action taken by BACPF when backing up to $NULL is to generate a listing of the file base.

Omission of this parameter causes the current value for this parameter to remain the same. The initial default for this parameter is UNSPECIFIED.

INCLUDE_ARCHIVE_INFORMATION or *IAI*

Reserved.

INCLUDE_DATA or *ID*

Reserved.

Remarks

- When using this subcommand, it is recommended that you always specify parameter names rather than depending upon positional placement of parameters. This is because it is anticipated that additional parameters will be added at a future date.
- For more information, see the NOS/VE System Usage manual.

SET_LIST_OPTIONS BACPF Subcommand

Purpose Specifies the information that is written to the list file by subsequent subcommands.

Format

SET_LIST_OPTIONS or
SET_LIST_OPTION or
SETLO
FILE_DISPLAY_OPTIONS=list of keyword
CYCLE_DISPLAY_OPTIONS=list of keyword
DISPLAY_EXCLUDED_ITEMS=boolean
STATUS=status variable

Parameters

FILE_DISPLAY_OPTIONS or *FILE_DISPLAY_OPTION* or *FDO*

Selects the data to be displayed with the file name. Options are:

ACCOUNT (A)

Displays the account name.

PROJECT (P)

Displays the project name.

NONE

Displays only the file name.

ALL

Displays the account and project name.

If the FILE_DISPLAY_OPTION parameter is omitted, NONE is selected.

CYCLE_DISPLAY_OPTIONS or *CYCLE_DISPLAY_OPTION* or *CDO*

Selects the data to be displayed for each cycle backed up or restored. The cycle number and whether the cycle was excluded is also displayed. Options are:

CREATION_DATE_TIME (CDT)

Displays the date and time the cycle was created.

ACCESS_DATE_TIME (ADT)

Displays the date and time the cycle was last accessed.

MODIFICATION_DATE_TIME (MDT)

Displays the date and time the cycle was last modified.

EXPIRATION_DATE (ED)

Displays the expiration date of the cycle.

ACCESS_COUNT (AC)

Displays the number of accesses to the cycle.

SIZE (S)

Displays the size of the cycle in bytes.

RECORDED_VSN (RVSN)

Displays all mass storage volumes on which the cycle resides.

GLOBAL_FILE_NAME (GFN)

Displays the internally generated global file name. This name is neither backed up nor restored.

NONE

Displays the cycle number.

ALL

Selects all of the display options.

If the CYCLE_DISPLAY_OPTION parameter is omitted, the MODIFICATION_DATE_TIME and SIZE options are selected.

DISPLAY_EXCLUDED_ITEMS or *DISPLAY_EXCLUDED_ITEM* or *DEI*

Determines whether or not excluded catalogs, files, and cycles are displayed on the list file. TRUE causes the identification of all excluded catalogs, files, and cycles to be displayed. If FALSE is specified, excluded items are not displayed. If omitted, TRUE is assumed.

Remarks For more information, see the NOS/VE System Usage manual.

CHANGE_KEYED_FILE and CREATE_KEYED_FILE 8

CHANGE_KEYED_FILE and CREATE_KEYED_FILE 8

CHANGE_KEYED_FILE Command

Purpose Begins a CHANGE_KEYED_FILE utility session.

Format

CHANGE_KEYED_FILE or
CHANGE_KEYED_FILES or
CHAKF
INPUT=file
OUTPUT=file
STATUS=status variable

Parameters

INPUT or **I**

File path of an existing keyed file. If an output file is specified, the input file is opened and copied to the output file and then closed.

This parameter is required.

OUTPUT or *O*

File path of the keyed file to which the input keyed file is copied. The output file must be a duplicate of the input file. If the output file does not exist, the command creates it.

If an output file is specified, only the output file is changed. If OUTPUT is omitted, the input file is changed.

Remarks

- The command utility prompt is:

```
chakf/
```

In response to the chakf/ prompt, you can enter SCL commands and any of these subcommands:

ADD_RECORDS
REPLACE_RECORDS
COMBINE_RECORDS
EXTRACT_RECORDS
DELETE_RECORDS
CREATE_NESTED_FILE
SELECT_NESTED_FILE
DELETE_NESTED_FILE

DISPLAY_NESTED_FILE
CREATE_ALTERNATE_INDEXES
HELP
QUIT

- All subcommands in the session apply to the currently selected nested file. The initially selected nested file is $MAIN_FILE. The nested file selection can be changed by a CREATE_NESTED_FILE or SELECT_NESTED_FILE subcommand.

- If the existing keyed file or a new nested file to be created uses a user-defined collation table, hashing procedure, or compression procedure, the object library containing the compiled table or procedure must be in the program library list before the Change_Keyed_File session begins.

  To add one or more object libraries to the program library list, use the ADD_LIBRARIES parameter on a SET_PROGRAM ATTRIBUTES command. For example:

  ```
  set_program_attributes, add_library=$user.hash_library
  ```

- For more information, see the NOS/VE Advanced File Management Usage manual.

Examples The following session copies an existing keyed file and then ends.

```
/change_keyed_file, input=$user.existing_keyed_file, ..
../output=$user.new_keyed_file
chakf/quit
/
```

CREATE_KEYED_FILE Command

Purpose Begins a CREATE_KEYED_FILE utility session.

Format

CREATE_KEYED_FILE or
CREATE_KEYED_FILES or
CREKF
OUTPUT=file
STATUS=status variable

Parameters

OUTPUT or O

File path of the keyed file to be created. The keyed-file attributes must already be specified by SET_FILE_ATTRIBUTES commands.

This parameter is required.

The minimum attributes that must be defined are KEY_LENGTH and MAXIMUM_RECORD_LENGTH. If the FILE_ORGANIZATION is omitted, Create_Keyed_File creates an indexed-sequential file.

Remarks

- The command utility prompt is:

  ```
  crekf/
  ```

  In response to the crekf/ prompt, you can enter SCL commands and any of these subcommands:

  ADD_RECORDS
  REPLACE_RECORDS
  COMBINE_RECORDS
  EXTRACT_RECORDS
  DISPLAY_RECORDS
  DELETE_RECORDS
  CREATE_NESTED_FILE
  SELECT_NESTED_FILE
  DELETE_NESTED_FILE
  DISPLAY_NESTED_FILE
  CREATE_ALTERNATE_INDEXES
  HELP
  QUIT

- The new keyed file is created with one nested file, named $MAIN_FILE. It is the initially selected nested file and all subcommands apply to it until a CREATE_NESTED_FILE or SELECT_NESTED_FILE subcommand selects another nested file.

- If any nested file in the new keyed file uses a user-defined collation table, hashing procedure, or compression procedure, the object library containing the compiled table or procedure must be in the program library list before the Create_Keyed_File session begins.

  To add one or more object libraries to the program library list, use the ADD_LIBRARIES parameter on a SET_PROGRAM_ATTRIBUTES command. For example:

```
set_program_attributes, add_library=$user.hash_library
```

- If you specify DIRECT_ACCESS as the FILE_ORGANIZATION attribute on the SET_FILE_ATTRIBUTES command, but omit the INITIAL_HOME_BLOCK_COUNT attribute, CREATE_KEYED_FILE prompts you for calculation of the INITIAL_HOME_BLOCK_COUNT.

- For more information, see the NOS/VE Advanced File Management Usage manual.

Examples This CREATE_KEYED_FILE example defines the file $USER.INDEXED SEQUENTIAL_FILE with the SET_FILE_ATTRIBUTES command and then creates it.

```
/set_file_attributes, file=$user.indexed_sequential_file ..
../file_organization=indexed_sequential ..
../maximum_record_length=32, minimum_record_length=14 ..
../key_length=14
/create_keyed_file, output=$user.indexed_sequential_file
crekf/
```

ADD_RECORDS
CHAKF and CREKF Subcommand

Purpose Adds records to the currently selected nested file.

Format

ADD_RECORDS or
ADD_RECORD or
ADDR
INPUT=list of any
SORT=boolean
ERROR_LIMIT=integer
STATUS=status variable

Parameters

INPUT or **I**

List of one or more files whose records are to be copied. You must have at least read access to the files.

This parameter is required.

SORT or *S*

Indicates whether the records are sorted before they are added to the file. (Sorting is recommended for better file performance.)

TRUE, ON, or YES

The records from the input file list are copied to a temporary file and sorted. Records for an indexed-sequential file are sorted by their primary-key value; records for a direct-access file are sorted by their hash value.

FALSE, OFF, or NO

The records are copied to a temporary file, but are not sorted.

If SORT is omitted, the default is TRUE.

ERROR_LIMIT or *EL*

Number of nonfatal errors required to force termination of the add (0 through [2**42-1]). A 0 sets an unlimited error limit.

If ERROR_LIMIT is omitted, 0 is used.

emarks For more information, see the NOS/VE Advanced File Management Usage manual.

xamples This Create_Keyed_File example creates the file $USER.INDEXED_SEQUENTIAL_FILE, adds the records of file $USER.ADD_RECORDS to it, and then displays the file.

```
/set_file_attributes ..
../file=$user.indexed_sequential_file ..
../file_organization=indexed_sequential ..
../maximum_record_length=32 ..
../minimum_record_length=14 ..
../key_length=14

/create_keyed_file ..
../output=$user.indexed_sequential_file
crekf/add_records input=$user.add_records

crekf/display_records count=all
                Display_Nested_File                    1986-02-17
NOS/VE Keyed File Utilities  1.2 85357                   11:19:36
File = :NVE.USER99.INDEXED_SEQUENTIAL_FILE.1
Display of records in $MAIN_FILE

 Byte: 0            ASCII: Everest       Asia          8848
 Byte: 0            ASCII: K2            Asia          8611
 Byte: 0            ASCII: Kilimanjaro   Africa        5895
 Byte: 0            ASCII: Matterhorn    Europe        4478
 Byte: 0            ASCII: McKinley      North America 6194
crekf/
```

COMBINE_RECORDS
CHAKF and CREKF Subcommand

Purpose Combines additional records with the records in the currently selected nested file.

Format

COMBINE_RECORDS or
COMBINE_RECORD or
COMR
INPUT=list of any
SORT=boolean
ERROR_LIMIT=integer
STATUS=status variable

Parameters

INPUT or **I**

List of one or more files whose records are to be copied. You must have at least read access to the files.

This parameter is required.

SORT or *S*

Indicates whether the input records are sorted before they are combined. (Sorting is recommended for better file performance.)

TRUE

The records from the input file list are copied to a temporary file and sorted. Records for an indexed-sequential file are sorted by their primary-key value; records for a direct-access file are sorted by their hash value.

FALSE

The records are copied to a temporary file, but are not sorted.

If SORT is omitted, the default is TRUE.

ERROR_LIMIT or *EL*

Number of nonfatal errors required to force termination of the combine (0 through [2**42-1]). A 0 sets an unlimited error limit.

If ERROR_LIMIT is omitted, 0 is used.

Remarks For more information, see the NOS/VE Advanced File Management Usage manual.

Examples This Create_Keyed_File example adds records that have a new primary key and replaces records that have an existing primary-key value.

```
/copy_keyed_file_add_file
Everest       Africa        8800
K2            Asia          8611
Kilimanjaro   Africa        5895

/copy_keyed_file combine_file
Everest       Asia          8848
Matterhorn    Europe        4478
McKinley      North America 6194

/create_keyed_file ..
../output=$user.indexed_sequential_file
crekf/add_records input-$user.add_file
crekf/combine_records input=$user.combine_file
crekf/display_records count=all
    Display_Nested_File                    1986-02-17
NOS/VE Keyed File Utilities  1.2 85357       12:01:46
File =:NVE.USER99.INDEXED_SEQUENTIAL_FILE.1
Display of records in $MAIN_FILE

 Byte: 0          ASCII: Everest       Asia          8848
 Byte: 0          ASCII: K2            Asia          8611
 Byte: 0          ASCII: Kilimanjaro   Africa        5895
 Byte: 0          ASCII: Matterhorn    Europe        4478
 Byte: 0          ASCII: McKinley      North America 6194
crekf/
```

CREATE_ALTERNATE_INDEXES CHAKF and CREKF Subcommand

Purpose Initiates execution of the CREATE_ALTERNATE_INDEXES command utility.

Format

CREATE_ALTERNATE_INDEXES or
CHANGE_ALTERNATE_INDEX or
CHANGE_ALTERNATE_INDEXES or
CHANGE_ALTERNATE_INDICES or
CREAI or
CREATE_ALTERNATE_INDEX or
CREATE_ALTERNATE_INDICES or
CHAAI
STATUS=status variable

Remarks

- The subutility prompt is:

  creai/

  In response to the creai/ prompt, you can enter NOS/VE commands and any of these subcommands:

  CREATE_KEY_DEFINITIONS

DISPLAY_KEY_DEFINITIONS
DELETE_KEY_DEFINITIONS
CANCEL_KEY_DEFINITIONS
APPLY_KEY_DEFINITIONS
HELP
QUIT

- For more information, see the NOS/VE Advanced File Management Usage manual.

Examples The following subutility session creates an alternate-key definition and then displays it.

```
  crekf/creat_alternate_indexes
  creai/create_key_definitions ..
  creai../key_name=alternate_key_1 ..
  creai../key_position=28 key_length=4
  creai/display_key_definitions display_options=all
  Display_Nested_File                                    1986-02-17
NOS/VE Keyed File Utilities  1.2 86034                   12:20:26
File = :NVE.INDEXED_SEQUENTIAL_FILE
Nested_File_Name

KEY_NAME                       POSITION LENGTH   TYPE        STATE
------------------------------ -------- -------- ----------- ----------------
ALTERNATE_KEY_1                      28        4  uncollated  creation pending
 Duplicate_Key_Value           : not_allowed
 Null_Suppression              : no
================================================================================

RECORD 1 .......(in ascii) :E v e r e s t                A s i a
                ( in hex ) :4576657265737420202020202020417369612020202020

ALTERNATE_KEY_1            :

                (in ascii) :        8 8 4 8
                ( in hex ) :2020202038383438
                           >        U_U_U_U_
creai/
```

CREATE_NESTED_FILE CHAKF and CREKF Subcommand

Purpose Creates and selects a new nested file.

Format

CREATE_NESTED_FILE or
CRENF
NAME=name
KEY_LENGTH=integer
KEY_POSITION=integer
KEY_TYPE=keyword
MAXIMUM_RECORD_LENGTH=integer
COLLATE_TABLE_NAME=name
COMPRESSION_PROCEDURE_NAME=list of any or

keyword
DATA_PADDING=integer
DYNAMIC_HOME_BLOCK_SPACE=boolean
EMBEDDED_KEY=boolean
FILE_ORGANIZATION=keyword
HASHING_PROCEDURE_NAME=list of any or *keyword*
INDEX_PADDING=integer
INITIAL_HOME_BLOCK_COUNT=integer
LOADING_FACTOR=integer
MINIMUM_RECORD_LENGTH=integer
RECORDS_PER_BLOCK=integer
RECORD_TYPE=keyword
STATUS=status variable

Parameters

NAME or **N**

Name of the new nested file. It must be unique in the keyed file.

This parameter is required.

KEY_LENGTH or **KL**

Primary-key length in bytes (for integer keys, 1 through 8; for character keys from 1 through 255).

This parameter is required.

KEY_POSITION or *KP*

Position of the leftmost byte of the primary key (specified only if the key is embedded). The byte positions in a record are numbered from the left, beginning with 0.

If KEY_POSITION is omitted, the default is 0.

KEY_TYPE or *KT*

Primary key type.

UNCOLLATED or UC

Key values ordered byte-by-byte according to the ASCII collating sequence.

INTEGER or I

Key values ordered numerically as integer values.

COLLATED or C

Key values ordered byte-by-byte according to the collating sequence specified by the COLLATE_TABLE_NAME parameter (invalid if FILE_ORGANIZATION=DIRECT_ACCESS).

If KEY_TYPE is omitted, the default is UNCOLLATED.

MAXIMUM_RECORD_LENGTH or **MAXRL**

Maximum number of bytes of data in a record (1 through 65497).

This parameter is required.

COLLATE_TABLE_NAME or *CTN*

Name of the collating sequence used to sort the primary key (indexed-sequential files only).

This parameter is required if the KEY_TYPE is COLLATED.

COMPRESSION_PROCEDURE_NAME or *CPN*

Name of the optional data compression or encryption procedure used with the nested file. The name can be either the name of the system-defined compression procedure (AMP$RECORD_COMPRESSION or the name of an entry point in the current program library list.

If COMPRESSION_PROCEDURE_NAME is omitted, the nested file does not use a compression procedure.

DATA_PADDING or *DP*

Percentage of data block space left empty when the indexed-sequential file is created (integer from 0 through 99).

The percentage must allow for storage of at least one maximum-length record per block.

If DATA_PADDING is omitted, the default is 0.

DYNAMIC_HOME_BLOCK_SPACE or *DHBS*

This parameter is reserved for future use. Its default value is FALSE.

EMBEDDED_KEY or *EK*

Indicates whether the primary-key value is embedded in the record data.

TRUE, ON, or YES

Primary-key value is embedded in the record data.

FALSE, OFF, or NO

Primary-key value is not part of the record data.

If EMBEDDED_KEY is omitted, the default is TRUE.

FILE_ORGANIZATION or *FO*

Keyed-file structure used.

INDEXED_SEQUENTIAL or IS

Data records accessed by searching for the primary-key value in a hierarchical index.

DIRECT_ACCESS or DA

Data record block accessed directly by hashed primary-key value.

If FILE_ORGANIZATION is omitted, the default is INDEXED_SEQUENTIAL.

HASHING_PROCEDURE_NAME or *HPN*

Name of the hashing procedure to be executed for the direct-access file.

If HASHING_PROCEDURE_NAME is omitted, the default is the system-provided hashing procedure (named AMP$SYSTEM_HASHING_PROCEDURE).

INDEX_PADDING or *IP*

Percentage of index block space left empty when the indexed-sequential file is created (integer from 0 through 99).

The percentage must allow for storage of at least one index record per block. (The length of an index record is the key length plus 4.)

If INDEX_PADDING is omitted, the default is 0.

INITIAL_HOME_BLOCK_COUNT or *IHBC*

Number of home blocks to be created in the direct-access file (1 through 2**31-1).

This parameter is required when FILE_ORGANIZATION=DIRECT_FILE_ORGANIZATION ACCESS.

LOADING_FACTOR or *LF*

Percentage of file space used when the direct-access file is created (no more than 90%).

If an initial home block count is specified, the loading factor is ignored. Otherwise, if LOADING_FACTOR is omitted, the default is 75%.

MINIMUM_RECORD_LENGTH or *MINRL*

Minimum number of bytes of data in a record (0 through 65497).

The minimum record length for a fixed-length record is the same as its maximum record length. The default minimum record length for variable-length records with an embedded key is the sum of the key_position and the key_length. Otherwise, the default minimum record length is 0.

RECORDS_PER_BLOCK or *RPB*

Reserved.

RECORD_TYPE or *RT*

Record type.

FIXED or F

Fixed-length records.

VARIABLE or V

Variable-length records.

UNDEFINED or U

Variable-length records.

If RECORD_TYPE is omitted, the default is UNDEFINED.

Remarks For more information, see the NOS/VE Advanced File Management Usage manual.

Examples This Create_Keyed_File example creates a new nested file NESTED_FILE_1 and then displays the newly created file.

```
crekf/create_nested_file name=nested_file_1 ..
crekf../maximum_record_length=32, key_length=14 ..
crekf../file_organization=indexed_sequential
crekf/display_nested_file
   Display_Nested_File                          1986-02-17
NOS/VE Keyed File Utilities 1.2 85357             12:42:49
File = :NVE.INDEXED_SEQUENTIAL_FILE

List of Nested Files for file INDEXED_SEQUENTIAL_FILE
   NESTED_FILE_1              (currently selected nested file)
   $MAIN_FILE
```

DELETE_NESTED_FILE
CHAKF and CREKF Subcommand

Purpose Deletes one or more nested files.

Format **DELETE_NESTED_FILE** or **DELNF**

NAMES=list of name

STATUS=status variable

Parameters **NAMES** or **NAME** or **N**

List of one or more nested files to be deleted.

This parameter is required.

Remarks

- You cannot delete the currently selected nested file or $MAIN_FILE.
- To delete the currently selected nested file, select another nested file first using the SELECT_NESTED_FILE subcommand and then issue the DELETE_NESTED_FILE subcommand.
- To display the names of the nested files, enter a DISPLAY_NESTED_FILE subcommand.
- For more information, see the NOS/VE Advanced File Management Usage manual.

Examples This Create_Keyed_File example displays the list of nested files and then deletes the nested file NESTED_FILE_2.

```
            crekf/display_nested_file
                Display_Nested_File                    1986-02-17
NOS/VE Keyed File Utilities  1.2 85357                   12:50:12
File = :NVE.INDEXED_SEQUENTIAL_FILE

List of Nested Files for file INDEXED_SEQUENTIAL_FILE
    NESTED_FILE_1              (currently selected nested file)
    NESTED_FILE_2
    $MAIN_FILE

crekf/delete_nested_file name=nested_file_2
crekf/display_nested_file
                Display_Nested_File                    1986-02-17
NOS/VE Keyed File Utilities  1.2 85357                   12:52:02
File =:NVE.INDEXED_SEQUENTIAL_FILE

List of Nested Files for file INDEXED_SEQUENTIAL_FILE
    $MAIN_FILE                 (currently selected nested file)
    NESTED_FILE_1
```

DELETE_RECORDS CHAKF and CREKF Subcommand

Purpose Deletes records from the currently selected nested file.

Format

DELETE_RECORDS or
DELETE_RECORD or
DELR
KEYS=range of any
COUNT=integer or *keyword*
VETO=boolean
STATUS=status variable

Parameters *KEYS* or *KEY* or *K*

Optional range of primary-key values to be deleted. The range may be specified as either:

1. Two primary-key values separated by two periods (..). (such as 'KEY1'..'KEY2'). The first key value must be less than the second. (Valid only for indexed-sequential files.)

2. One primary-key value specifying the beginning of the range. The number of records in the range is specified by the COUNT parameter.

The keywords $FIRST_KEY and $LAST_KEY can specify the lowest and highest key values, respectively, in an indexed-sequential file.

If KEYS is omitted, the range of records to be deleted begins with the first record in the nested file.

COUNT or *C*

Number of records to be deleted (0 through 2**42-1 or, to delete all records, the keyword ALL or A).

If a range is specified by the KEYS parameter, the COUNT value limits the number of records deleted.

If COUNT is omitted, but KEYS is specified, the default count the number of records in the specified range. Otherwise, the default is 1.

VETO or *V*

Indicates whether the interactive user must confirm each deletion.

TRUE

Each record to be deleted is displayed with the prompt Okay to delete?==>.

FALSE

All specified records are deleted.

If VETO is omitted, the default is FALSE.

The possible responses to the veto prompt are:

YES or Y

Delete the record.

NO or N

Do not delete the record.

ALL or A

Delete the rest of the records without prompts.

QUIT or Q

Stop without deleting any more records.

HEX or H

Redisplays the record in hexadecimal and reissues the prompt.

Remarks For more information, see the NOS/VE Advanced File Management Usage manual.

Examples This Create_Keyed_File example deletes a record in the currently selected nested file.

```
crekf/delete_records, keys='Matterhorn'..'McKinley' ..
crekf../count=2, veto=true
Byte: 0              ASCII: Matterhorn     Europe        4478
Okay to delete:  ==>Yes
Byte: 0              ASCII: McKinley       North America 6194
Okay to delete:  ==>No
--INFORMATIVE AA 501285-- As requested by the user, this record was not
deleted.
--INFORMATIVE AA 501285-- The Delete_Records subcommand of
CREATE_KEYED_FILE deleted 1 record from nested file $MAIN_FILE in file
:NVE.INDEXED_SEQUENTIAL_FILE.
crekf/
```

DISPLAY_NESTED_FILE CHAKF and CREKF Subcommand

Purpose Displays the nested file definitions and the alternate-key names and number of records in each nested file.

Format **DISPLAY_NESTED_FILE** or **DISNF**

NAMES=list of name or *keyword*
OUTPUT=file
DISPLAY_OPTIONS=list of keyword
STATUS=status variable

Parameters *NAMES* or *NAME* or *N*

List of one or more names of nested files to be displayed or the keyword ALL to display all nested files in the file.

If NAMES is omitted, the default is ALL.

OUTPUT or *O*

File to which the display is written. The file must be a sequential file.

If OUTPUT is omitted, the default file is $OUTPUT.

DISPLAY_OPTIONS or *DISPLAY_OPTION* or *DO*

List of one or more keywords indicating the type of information to be displayed.

DEFINITIONS or DEFINITION or D

Nested-file definitions.

KEY_NAMES or KEY_NAME or K

Names of the alternate keys in each nested file.

NAMES or NAME or N

Nested-file names.

RECORD_COUNTS or RECORD_COUNT or RC

Number of records in each nested file.

If DISPLAY_OPTION is omitted, the default is NAMES.

Remarks

- The currently selected nested file is marked as such in the list of nested files.
- For more information, see the NOS/VE Advanced File Management Usage manual.

Examples

This Create_Keyed_File example displays the default nested file ($MAIN_FILE) with the DISPLAY_OPTIONS parameter set to ALL. No alternate keys have been defined.

```
            crekf/display_nested_file/Display_options=all
               Display_Nested_File                        1986-02-17
NOS/VE Keyed File Utilities  1.2 86034                      12:59:58
File = :NVE.INDEXED_SEQUENTIAL_FILE

$MAIN_FILE                        (currently selected nested file)
Record_Count                      : 3

Nested_File_Definitions
  Compression_Procedure_Name      : none
  Embedded_Key                    : yes
  Key-Position                    : 0
  Key-Length                      : 14
  Maximum_Record_Length           : 32
  Minimum_Record_Length           : 32
  Record_Type                     : undefined
  File_Organization               : indexed_sequential
  Key_Type                        : uncollated
  Collate_Table_Name              :
  Data_Padding                    : 0
  Index Padding                   : 0
```

DISPLAY_RECORDS
CHAKF and CREKF Subcommand

Purpose Displays records in the currently selected nested file.

Format

DISPLAY_RECORDS or
DISPLAY_RECORD or
DISR
OUTPUT=file
KEYS=range of any
COUNT=integer or *keyword*
DISPLAY_OPTION=keyword
STATUS=status variable

Parameters

OUTPUT or *O*

File to which the display is written. The file must be a sequential file for which you have append access.

If OUTPUT is omitted, $OUTPUT is the default.

KEYS or *KEY* or *K*

Optional range of primary-key values to be displayed. The range may be specified as either:

1. Two primary-key values separated by two periods (..). (such as 'KEY1'..'KEY2'). The first key value must be less than the second. (Valid only for indexed-sequential files.)

2. One primary-key value specifying the beginning of the range. The number of records in the range is specified by the COUNT parameter.

The keywords $FIRST_KEY and $LAST_KEY can specify the lowest and highest key values, respectively, in an indexed-sequential file.

If KEYS is omitted, the range of records to be displayed begins with the first record in the nested file.

COUNT or *C*

Number of records to be displayed (0 through 2**42-1 or, to display all records, the keyword ALL or A).

If a range is specified by the KEYS parameter, the COUNT value limits the number of records displayed.

If COUNT is omitted, but KEYS is specified, the default count is the number of records in the specified range. Otherwise, the default is 1.

DISPLAY_OPTION or *DISPLAY_OPTIONS* or *DO*

List of one or more keywords indicating the representation used to display records.

ASCII

ASCII characters.

HEX or H

Hexadecimal digits.

BOTH

Both ASCII characters and hexadecimal digits.

ALTERNATE_KEY_DEFINITION or AKD or ALL

Both ASCII and hexadecimal representation with alternate-key values marked.

If DISPLAY_OPTION is omitted, the default is ASCII.

Remarks

- The ALTERNATE_KEY_DEFINITION display shows the record contents in ASCII characters and hexadecimal digits with the alternate-key values underscored.
- For more information, see the NOS/VE Advanced File Management Usage manual.

Examples The following session displays a range of records showing both ASCII and hexadecimal representations.

```
crekf/display_records display_option=both ..
crekf../keys='Everest'..'Kilimanjaro'
    Display_Nested_File      1986-04-23
NOS/VE Keyed File Utilities 1.2 86099      15:08:18
File = :NVE.USER99.INDEXED_SEQUENTIAL_FILE.1
Display of records in $MAIN_FILE for:

  COUNT: all
FIRST_KEY: Everest
 LAST_KEY: Kilimanjaro
Byte: 0         ASCII: E v e r e s t                    A s i a
Byte: 0(16        HEX: 457665726573742020202020202041736961202020202020
Byte: 25        ASCII:        8 8 4 8
Byte: 19(16)      HEX: 20202038303438
Byte: 0         ASCII: K 2                              A s i a
Byte: 0(16)       HEX: 4B322020202020202020202020204173696120202020202020
Byte: 25        ASCII:        8 6 1 1
Byte: 19(16)      HEX: 20202038363131
Byte: 0         ASCII: K i l i m a n j a r o      A f r i c a
Byte: (16)        HEX: 4B696C696D616E6A61726F2020204166726963612020202020
Byte: 25        ASCII:        5 8 9 5
Byte: 19(16)      HEX: 20202035383935
crekf/
```

EXTRACT_RECORDS
CHAKF and CREKF Subcommand

Purpose Copies records from the currently selected nested file.

Format

EXTRACT_RECORDS or
EXTRACT_RECORD or
EXTR
OUTPUT=list of any
KEYS=range of any
COUNT=integer or *keyword*
ERROR_LIMIT=integer
STATUS=status variable

Parameters

OUTPUT or *O*

File to which records are copied. You must have at least append access to the file.

If OUTPUT is omitted, the default is $OUTPUT.

KEYS or *KEY* or *K*

Optional range of primary-key values of the records to be copied. The range may be specified as either:

1. Two primary-key values separated by two periods (..) (such as 'KEY1'..'KEY2'). The first key value must be less than the second. (Valid only for indexed-sequential files.)

2. One primary-key value specifying the beginning of the range. The number of records in the range is specified by the COUNT parameter.

The keywords $FIRST_KEY and $LAST_KEY can specify the lowest and highest key values, respectively, in an indexed-sequential file.

If KEYS is omitted, the range of records to be copied begins with the first record in the nested file.

COUNT or *C*

Number of records to be copied (0 through 2**42-1 or, to copy all records, the keyword ALL or A).

If a range is specified by the KEYS parameter, the COUNT value limits the number of records copied.

If COUNT is omitted, but KEYS is specified, the default count is the number of records in the specified range. Otherwise, the default is 1.

ERROR_LIMIT or *EL*

Reserved.

Remarks

- Records are extracted only from the currently selected nested file.

- For more information, see the NOS/VE Advanced File Management Usage manual.

HELP
CHAKF and CREKF Subcommand

Purpose Displays information about utility subcommands.

Format

HELP or
HEL
SUBJECT=string
MANUAL=file
STATUS=status variable

Parameters

SUBJECT or *S*

Index topic to be located in the online manual.

If SUBJECT is omitted, the HELP subcommand lists the names of the utility subcommands.

MANUAL or *M*

Online manual file.

If MANUAL is omitted, the default is $SYSTEM.MANUALS.AFM.

Remarks

- If you enter a topic that is not in the manual index, a message appears telling you that the topic could not be found.
- The default manual, $SYSTEM.MANUALS.AFM, contains the online version of the NOS/VE Advanced File Management Usage manual, as provided with the NOS/VE system.
- If your terminal is defined for screen applications, the online manual is displayed in screen mode.

  To leave the online manual, use QUIT. To get help on reading the online manual, use HELP.
- For more information, see the NOS/VE Advanced File Management Usage manual.

QUIT
CHAKF and CREKF Subcommand

Purpose Ends the utility session and closes the output file.

Format

QUIT or
QUI
STATUS=status variable

Remarks For more information, see the NOS/VE Advanced File Management manual.

REPLACE_RECORDS
CHAKF and CREKF Subcommand

Purpose Replaces existing records in the currently selected nested file.

Format

REPLACE_RECORDS or
REPLACE_RECORD or
REPR
INPUT=list of any
SORT=boolean
ERROR_LIMIT=integer
STATUS=status variable

Parameters

INPUT or I

List of one or more files whose records are to replace the corresponding records already in the keyed file. You must have at least read access to the input files.

This parameter is required.

SORT or *S*

Indicates whether the records are sorted before they are copied to the file. (Sorting is recommended for better file performance.)

TRUE, ON, or YES

The records from the input file list are copied to a temporary file and sorted. Records for an indexed-sequential file are sorted by their primary-key value; records for a direct-access file are sorted by their hash value.

FALSE, OFF, or NO

The records are copied to a temporary file, but are not sorted.

If SORT is omitted, the default is TRUE.

ERROR_LIMIT or *EL*

Number of nonfatal errors required to force termination of the replace (0 through [2**42-1]). A 0 sets an unlimited error limit.

If ERROR_LIMIT is omitted, 0 is used.

Remarks For more information, see the NOS/VE Advanced File Management Usage manual.

Examples This Create_Keyed_File example replaces records in file $USER.INDEXED_SEQUENTIAL_FILE that have the same primary key.

```
/copy_keyed_file $user.add_file
Everest       Africa        8800
K2            Asia          8611
Kilimanjaro   Africa        5895

/copy_keyed_file $user.replace_file
Everest       Asia          8848

/create_keyed_file ..
../output=$user.indexed_sequential_file
crekf/add_records input=$user.add_file
crekf/replace_records input=$user.replace_file
crekf/display_records count=all
    Display_Nested_file                          1986-02-17
NOS/VE Keyed File Utilities  1.2 85357             13:19:24
File = :NVE.USER99.INDEXED_SEQUENTIAL_FILE.1
Display of records in $MAIN_FILE

 Byte: 0            ASCII: Everest       Asia          8848
 Byte: 0            ASCII: K2            Asia          8611
 Byte: 0            ASCII: Kilimanjaro   Africa        5895
crekf/
```

SELECT_NESTED_FILE CHAKF and CREKF Subcommand

Purpose Selects the nested file to which subsequent subcommands are to apply.

Format

SELECT_NESTED_FILE or
SELNF
NAME=name
STATUS=status variable

Parameters NAME or N

Name of an existing nested file. To select the default nested file, specify $MAIN_FILE.

This parameter is required.

Remarks For more information, see the NOS/VE Advanced File Management manual.

CREATE_ALTERNATE_INDEXES 9

CREATE_ALTERNATE_INDEXES Command

Purpose Initiates execution of the CREATE_ALTERNATE_INDEXES command utility. The utility can create, delete, and display alternate-key definitions in a keyed file.

Format

CREATE_ALTERNATE_INDEXES or
CREATE_ALTERNATE_INDEX or
CREATE_ALTERNATE_INDICES or
CREAI
INPUT=list of any
STATUS=status variable

Parameters

INPUT or **I**

Keyed file to be processed by the utility. The file permissions required depend on the subcommands entered during the utility as described in the Remarks. This parameter is required.

To specify a nested file, first specify the file reference and then the nested-file name, enclosed in parentheses.

Remarks

- The command utility prompt is:

```
creai/
```

- In response to the creai/ prompt, you can enter NOS/VE commands and any of these subcommands:

  QUIT
  DISPLAY_KEY_DEFINITIONS
  CREATE_KEY_DEFINITION
  DELETE_KEY_DEFINITION
  CANCEL_KEY_DEFINITIONS
  APPLY_KEY_DEFINITIONS

- The CREATE_ALTERNATE_INDEXES utility creates the specified keyed file if:
  - The file does not exist and,
  - A SET_FILE_ATTRIBUTES command has specified the KEY_LENGTH and MAXIMUM_RECORD_LENGTH attributes for the file.

  If the SET_FILE_ATTRIBUTES command defining the new file omits an attribute, the default attribute value is used. However, if it omits the FILE_ORGANIZATION attribute, indexed-sequential organization is used.

- The CREATE_ALTERNATE_INDEXES command does not check your file permissions. The subcommands you enter in the utility session check that you have the required permissions to do the operation.

  To display key definitions, you must have at least read permission. To create, delete, cancel, or apply key definitions, you must have at least three permissions: append, modify, and shorten.

- For more information, see the NOS/VE Advanced File Management Usage manual.

Examples This command begins a utility session that displays the alternate key definitions of keyed file $USER.IS_FILE.

```
/create_alternate_indexes input=$user.is_file
creai/display_key_definitions key_names=all display_options=brief
       Display_Key_Definitions        NOS/VE Keyed File Utilities 1.1
File = :NVE.USER99.IS_FILE

 KEY NAME                  POSITION    LENGTH   TYPE         STATE
 ------------------------ --------    ------   ----------   --------------
 ALTERNATE_KEY_1                 0        10   uncollated   Exists in file
creai/quit      "The APPLY_KEY_DEFINITIONS parameter is not required here"
                "because no creation or deletion requests are pending."
```

ADD_PIECE
CREKD Subcommand

Purpose Defines a piece of a concatenated key within a CREATE_KEY_DEFINITION utility session.

Format

ADD_PIECE or
ADDP
POSITION=integer
LENGTH=integer
TYPE=keyword
STATUS=status variable

Parameters

POSITION or **P** or **KEY_POSITION** or **KP**

Byte position in the record at which the piece begins. The byte positions are numbered from the left, beginning with 0. The maximum byte position is 65496. This parameter is required.

LENGTH or **L** or **KEY_LENGTH** or **KL**

Number of bytes in the piece. The maximum length is 255 bytes. The piece must be within the minimum record length unless sparse-key control is used. This parameter is required.

TYPE or *T* or *KEY_TYPE* or *KT*

Type of the piece.

INTEGER (I)

Integer key ordered numerically.

UNCOLLATED (UC or U)

Character key ordered byte-by-byte according to the ASCII collating sequence.

COLLATED (C)

Character key ordered byte-by-byte according to the collation table specified by the COLLATE_TABLE_NAME parameter on the CREATE_KEY_DEFINITION command.

The default key type is UNCOLLATED.

Remarks

- The utility is initiated in response to a CREATE_KEY_DEFINITION subcommand that specifies the CONCATENATED_PIECES=TRUE parameter.
- To end concatenated-key specification, enter the QUIT subcommand for the CREATE_KEY_DEFINITION utility.
- You must enter an ADD_PIECE subcommand for each piece to be concatenated to the first piece to define a concatenated key. The first piece is defined by the KEY_LENGTH, KEY_POSITION, and KEY_TYPE parameters on the CREATE_KEY_DEFINITION command.
- A concatenated key can comprise from 2 through 64 pieces. The pieces are concatenated in the order that you enter the ADD_PIECE subcommands that define the pieces.
- For more information, see the NOS/VE Advanced File Management Usage manual.

Examples

This CREATE_ALTERNATE_INDEXES session defines an alternate key that concatenates the first, third and fifth bytes of the record in reverse order. It displays the definition and then cancels the request.

```
/create_alternate_index input=$user.is_file
creai/create_key_definition key_name=alternate_key_2 ..
creai../key_position=4 key_length=1 concatenated_pieces=yes
crekd/add_piece key_position=2 key_length=1
crekd/addp kp=0 kl=1
crekd/quit
creai/display_key_definitions
            Display_Key_Definitions      NOS/VE Keyed File Utilities 1.1
 File = .NVE.USER99.IS_FILE
 KEY NAME                        POSITION LENGTH TYPE       STATE
 ------------------------------- -------- ------ ---------- ----------------
 ALTERNATE_KEY_2                        4      1 uncollated Creation pending
                       piece b          2      1 uncollated
                       piece c          0      1 uncollated
   Duplicate_Key_Values        : not_allowed
   Null_Suppression            : no
================================================================================
RECORD 1 .......(in ascii) : T h i s   i s   t h e   f i r s t   r e c o r d
                ( in hex ) : 546869732069732074686520666972737420726563 6F72
ALTERNATE_KEY_2            : c_  b_  a_
                (in ascii) : d .
                ( in hex ) : 642E
                           >
creai/quit cancel
/
```

APPLY_KEY_DEFINITIONS CREAI Subcommand

Purpose Applies the pending alternate-key definition and deletion requests within a CREATE_ALTERNATE_INDEXES utility session.

Format

APPLY_KEY_DEFINITIONS or
APPLY_KEY_DEFINITION or
APPKD
ERROR_LIMIT=integer
STATUS=status variable

Parameters

ERROR_LIMIT or *EL*

Number of trivial (nonfatal) errors allowed for the apply operation (integer from 0 through 4398046511103 [2**42-1]).

A 0 value indicates no limit; 0 is the default value.

See Remarks for a description of apply error processing.

Remarks

- This CREATE_ALTERNATE_INDEXES subcommand applies all pending alternate-key creation and deletion requests to the file. It applies deletion requests first and then the creation requests.

- The ERROR_LIMIT file attribute value has no effect on keyed-file utility processing. This is done so that nonfatal errors (such as typing errors during interactive use) do not terminate the utility session.

  However, you can specify an error limit that applies to the apply operation only by specifying the ERROR_ LIMIT parameter.

- The two nonfatal (trivial) errors that an apply operation can detect result from improper record data, as follows:

  Duplicate_Key_Value

  The duplicate_key_value attribute of the alternate index being built is NOT_ALLOWED, but the apply operation finds an alternate-key value matching an alternate-key value already in the alternate index.

Sparse_Key_Beyond_EOR

The apply operation is building an alternate index that uses sparse-key control and it finds a record for which an alternate-key value should be included in the index except that the record is too short to provide a complete alternate-key value.

- APPLY_KEY_DEFINITIONS keeps a count of the number of times it detects a nonfatal (trivial) error. Each time it increments the count, it checks whether the count has reached the value specified on the ERROR_LIMIT parameter.

  - If the error limit is not yet reached, APPLY_KEY_DEFINITIONS performs the correction processing. for the condition as described later.

  - If the error limit is reached, APPLY_KEY_DEFINITIONS terminates with a fatal error. The fatal error returned depends on the last nonfatal error detected:

    For a Duplicate_Key_Value error, it returns AAE$DUPLICATE_KEY_LIMIT.

    For a Sparse_Key_Beyond_EOR error, it returns AAE$ERROR_LIMIT_EXCEEDED.

- Before terminating, APPLY_KEY_DEFINITIONS discards all alternate indexes it has built. (Deleted alternate indexes are not restored.)

- If APPLY_KEY_DEFINITIONS finds one or more nonfatal errors, but completes its processing before reaching the error limit, it returns a warning message.

- As correction processing for a sparse_key_beyond_EOR error, APPLY_KEY_DEFINITIONS does not enter an alternate-key value for the record in the alternate index it is building, even though the sparse-key character indicates that a value should be entered for the record.

- As correction processing for a Duplicate_Key_Value error, APPLY_KEY_DEFINITIONS changes the duplicate_key_values attribute of the alternate-key definition from NOT_ALLOWED to ORDERED_BY_

PRIMARY_KEY. It then discards the partially-built index and begins building the index again, ordering duplicate alternate-key values by their primary-key value.

- Entry of a pause-break character is ignored during application of alternate-key definitions.
- Entry of a terminate_break_character during application of alternate-key definitions returns a prompt to the terminal user, asking for confirmation.
- As described in the prompt, the terminal user should then enter a carriage return or any entry other than RUIN FILE (uppercase or lowercase) to continue the application of alternate-key definitions. Applied alternate-key definitions can be removed without harm to the file after the apply operation executes.
- A request to ruin the file is not recommended. No file operation can be performed on a ruined file; therefore, no data can be retrieved from the file.
- For more information, see the NOS/VE Advanced File Management Usage manual.

Examples

This CREATE_ALTERNATE_INDEXES session attempts to create and apply an alternate key. The attempt fails when it finds a duplicate alternate-key value because the alternate-key definition does not allow duplicate values and the error limit for the apply is 1.

```
/create_alternate_indexes input=$user.is_file
creai/create_key_definition key_name=alternate_key_6 ..
creai../key_position=5 key_length=10
creai/apply_key_definition error_limit=1
-- File :NVE.USER99.IS_FILE : begin creating labels for
alternate key definitions.
-- File :NVE.USER99.IS_FILE : finished creating labels for
alternate key definitions.
-- File :NVE.USER99.IS_FILE : begin collecting the alternate
key values from the file.
-- File :NVE.USER99.IS_FILE : AMP$APPLY_KEY_DEFINITIONS has
reached a file boundary: EOI.
-- File :NVE.USER99.IS_FILE : collecting of the alternate key
values completed.
-- File :NVE.USER99.IS_FILE : begin sorting the alternate key
values.
-- File :NVE.USER99.IS_FILE : sorting of the alternate key
values completed.
-- File :NVE.USER99.IS_FILE : begin building alternate key
indexes into the file.
-- File :NVE.USER99.IS_FILE : the ALTERNATE_KEY_6 index
is being built.
-- File :NVE.USER99.IS_FILE : Alternate key ALTERNATE_KEY_6
has been deleted.
--ERROR-- File :NVE.USER99.IS_FILE :
AMP$APPLY_KEY_DEFINITIONS encountered a duplicate key
and found that error limit had been reached.  Because
ERROR_LIMIT was involved, any new indexes were removed
(though deleted indexes are gone).  Had ERROR_LIMIT not
been reached, the key definition would have been
modified to allow duplicates.  The duplicate key values
relate to alternate key name = ALTERNATE_KEY_6, primary
key = 96070, alternate_key_value = John Smith.
-- FATAL-- File  :NVE.USER99.IS_FILE :
AMP$APPLY_KEY_DEFINITIONS :  the user-declared maximum
number of trivial errors has been recorded since the
last OPEN.
creai/quit
/
```

CANCEL_KEY_DEFINITIONS CREAI Subcommand

Purpose Removes a pending request to create or delete an alternate key within a CREATE_ALTERNATE_INDEXES utility session.

Format

CANCEL_KEY_DEFINITIONS or
CANCEL_KEY_DEFINITION or
CANKD
NAMES=list of name
STATUS=status variable

Parameters

NAMES or NAME or N or KEY_NAMES or KEY_NAME or KN

Pending requests to be canceled.

list of names

Cancel the requests for the listed alternate-key names.

ALL

Cancel all requests.

This parameter is required.

Remarks

- The CANCEL_KEY_DEFINITIONS subcommand can cancel creation and deletion requests only while they are pending.
- After a creation or deletion request is applied, the CANCEL_KEY_DEFINITIONS subcommand has no effect. To reverse the action of an APPLY_KEY_DEFINITIONS subcommand, you must issue new requests to delete the created alternate key or recreate the deleted alternate key.
- For more information, see the NOS/VE Advanced File Management Usage manual.

Examples

This CREATE_ALTERNATE_INDEXES session requests creation of an alternate key and deletion of another alternate key, cancels the creation request, and finally applies the deletion request.

```
/create_alternate_indexes input=$user.is_file
creai/create_key_definition key_name=alternate_key_4 ..
creai../key_position=5 key_length=2
creai/delete_key_definition key_name=alternate_key_1
creai/cancel_key_definition alternate_key_4
creai/quit apply
-- File :NVE.USER99.IS_FILE : begin deleting alternate key
definitions.
-- File :NVE.USER99.IS_FILE : Alternate key ALTERNATE_KEY_1
has been deleted.
-- File :NVE.USER99.IS_FILE : end deleting alternate key
definitions.
/
```

CREATE_KEY_DEFINITION CREAI Subcommand

Purpose Creates a pending alternate-key definition within a CREATE_ALTERNATE_INDEXES utility session.

Format

CREATE_KEY_DEFINITION or
CREKD
NAME=name
POSITION=integer
LENGTH=integer
TYPE=keyword
COLLATE_TABLE_NAME=name
DUPLICATE_KEY_VALUES=boolean or *keyword*
NULL_SUPPRESSION=boolean
SPARSE_KEY_CONTROL_POSITION=integer
SPARSE_KEY_CONTROL_CHARACTERS=string
SPARSE_KEY_CONTROL_EFFECT=keyword
REPEATING_GROUP_LENGTH=integer
REPEATING_GROUP_COUNT=integer or *keyword*
GROUP_NAME=name
CONCATENATED_PIECES=boolean
VARIABLE_LENGTH_KEY=string
STATUS=status variable

Parameters

NAME or N or KEY_NAME or KN

Name of the new alternate key. The name must follow the SCL naming rules. This parameter is required.

POSITION or P or KEY_POSITION or KP

Byte position within the record at which the alternate-key field begins. The byte positions are numbered from the left, beginning with 0. The maximum byte position is 65496. This parameter is required.

LENGTH or L or KEY_LENGTH or KL

Number of bytes in the alternate-key field. The maximum length is 255 bytes. The key field must be within the minimum record length (unless sparse key control is used). This parameter is required.

TYPE or *T* or *KEY_TYPE* or *KT*

Type of the alternate key.

INTEGER (I)

Integer key ordered numerically.

UNCOLLATED (UC or U)

Character key ordered byte-by-byte according to the ASCII collating sequence.

COLLATED (C)

Character key ordered byte-by-byte according to the collation table specified by the COLLATE_TABLE_NAME parameter.

If the KEY_TYPE parameter is omitted, the key type is UNCOLLATED.

COLLATE_TABLE_NAME or *CTN*

Name of the collation table used to order the alternate key if its key type is collated. The collation table can be for NOS/VE predefined collating sequence or a user-defined collating sequence.

If the file is an indexed-sequential file with a collated primary key, the collation table for the primary key is used as the default collation table for an alternate key.

Otherwise, you must specify a collation table for a collated alternate key.

DUPLICATE_KEY_VALUES or *DKV*

Indicates whether duplicate alternate-key values are allowed and, if so, how the duplicate values are ordered.

NOT_ALLOWED (NA), FALSE (OFF or NO)

No duplicate values are allowed for the alternate key.

ORDERED_BY_PRIMARY_KEY (OBPK), TRUE (ON or YES)

Duplicate values are allowed. Duplicates are accessed in order by their primary key.

FIRST_IN_FIRST_OUT (FIFO)

Duplicate values are allowed. Duplicates are accessed in the order the values were entered in the index.

If the DUPLICATE_KEY_VALUES parameter is omitted, no duplicate values are allowed.

NULL_SUPPRESSION or *NS*

Indicates whether null alternate-key values should be stored in the alternate index. (The null value is all zeros for integer keys, all blanks for the other key types.)

TRUE (ON or YES)

Null values are not included in the index.

FALSE (OFF or NO)

All values are included in the index.

If the NULL_SUPPRESSION parameter is omitted, all values, including nulls, are stored in the index.

SPARSE_KEY_CONTROL_POSITION or *SKCP*

Byte position of the sparse-key control character. The position must be within the minimum record length. The byte positions are numbered from the left, beginning with 0. The maximum byte position is 65496.

NOTE

The two parameters, SPARSE_KEY_CONTROL_POSITION and SPARSE_KEY_CONTROL_CHARACTERS, work together; they must either both be specified or both be omitted. If they are omitted, sparse-key control is not used for the alternate key.

SPARSE_KEY_CONTROL_CHARACTERS or *SKCC*

String containing the set of characters with which the sparse-key control character in each record is compared.

SPARSE_KEY_CONTROL_EFFECT or *SKCE*

Indicates whether a sparse-key control character match causes the alternate-key value to be included in or excluded from the alternate index.

INCLUDE_KEY_VALUE (IKV)

The alternate-key value is included in the alternate index.

EXCLUDE_KEY_VALUE (EKV)

The alternate-key value is excluded from the alternate index.

You can specify the SPARSE_KEY_EFFECT parameter only if you specify the SPARSE_KEY_POSITION and SPARSE_KEY_CHARACTERS parameters.

If the SPARSE_KEY_CONTROL_EFFECT parameter is omitted, INCLUDE_KEY_VALUE is used.

REPEATING_GROUP_LENGTH or *RGL*

Length, in bytes of the repeating group of fields. It is the distance from the beginning of an alternate-key value to the beginning of the next value for the same alternate key in the same record.

The group length range is from 1 through 65497.

If the REPEATING_GROUP_LENGTH parameter is omitted, the alternate key has no more than one value per record.

REPEATING_GROUP_COUNT or *RGC*

Indicates how many alternate-key values are in a record. (The alternate-key value is in a repeating group of fields.)

integer (1 through 65497)

Number of times the alternate key occurs in a record. The specified number of alternate-key values must occur within the minimum record length.

REPEAT_TO_END_OF_RECORD (RTEOR)

The alternate key repeats until the record ends. (An incomplete key at the end of the record is not used.)

You can specify the REPEATING_GROUP_COUNT parameter only if you specify the REPEATING_GROUP_LENGTH parameter.

If the REPEATING_GROUP_COUNT parameter is omitted, the alternate key repeats until the end of the record.

GROUP_NAME or *GN* or *KEY_GROUP_NAME* or *KGN*

Name of the key group for this key. The key-grouping feature is not currently implemented. The default value for the key-group name is the key name.

CONCATENATED_PIECES or *CONCATENATED_PIECE* or *CP*

Indicates whether the alternate key is a concatenated key.

TRUE (ON or YES)

The key is a concatenated key.

FALSE (OFF or NO)

The key is not a concatenated key.

If you specify CONCATENATED_PIECES=TRUE, the CREATE_KEY_DEFINITION command initiates the CREATE_KEY_DEFINITION subcommand utility. The utility prompt is crekd/ and it processes ADD_PIECE and QUIT subcommands.

If the CONCATENATED_PIECES parameter is omitted, the key is not a concatenated key.

VARIABLE_LENGTH_KEY or *VLK*

Indicates that the key is a variable_length key by specifying its set of delimiter characters. The set is specified as a string (0 through 256 characters, enclosed in apostrophes).

If the REPEATING_GROUP_LENGTH parameter is omitted, no more than one value for the key is taken from a record. The end of the value is marked by a delimiter character, by the end of the key field (KEY_LENGTH length), or by the end of the record, whichever occurs first after the KEY_POSITION.

If the REPEATING_GROUP_LENGTH parameter is specified, the record can contain more than one value for the key. Multiple key values are separated by one or more delimiter characters. The REPEATING_GROUP_COUNT parameter indicates whether the sequence of values continues to the end of the record or is limited to a fixed number of characters.

If VARIABLE_LENGTH_KEY is omitted, the alternate key has fixed-length values.

Remarks

- The CREATE_KEY_DEFINITION subcommand defines an alternate key but does not apply the definition to the file. The definition remains pending until it is either applied or canceled.
- A definition is applied by either an APPLY_KEY_DEFINITIONS subcommand or an APPLY_KEY_DEFINITIONS=YES parameter on the QUIT subcommand. It is canceled by a CANCEL_KEY_DEFINITIONS subcommand or an APPLY_KEY_DEFINITIONS=NO parameter on the QUIT subcommand.
- The REPEATING_GROUP_LENGTH and the VARIABLE_LENGTH_KEY parameters cannot be specified with either the CONCATENATED_PIECES parameter or the DUPLICATE_KEY_VALUES=FIRST_IN_FIRST_OUT parameter.
- If the alternate-key definition defines a collated key, CREATE_KEY_DEFINITIONS searches for the collation-table name as an entry point in the object libraries in the program-library list.
- You must set the program-library list before you enter the utility. You cannot change the object libraries searched from within the utility session.
- The following command adds an object library to the program-library list:

  ```
  /set_program_attributes add_library=file_reference
  ```

- For more information, see the NOS/VE Advanced File Management Usage manual.

Examples This CREATE_ALTERNATE_INDEXES utility session creates and applies an alternate-key definition to file $USER.IS_FILE.

```
/create_alternate_index  input=$user.is_file
creai/create_key_definition  key_name=alternate_key_1 ..
creai../key_position=0  key_length=10
creai/quit  apply_key_definition
-- File :NVE.USER99.IS_FILE : begin creating labels for
alternate key definitions.
-- File :NVE.USER99.IS_FILE : finished creating labels for
alternate key definitions.
-- File :NVE.USER99.IS_FILE : begin collecting the alternate
key values from the file.
-- File :NVE.USER99.IS_FILE : AMP$APPLY_KEY_DEFINITIONS has
reached a file boundary: EOI .
-- File :NVE.USER99.IS_FILE : collecting of the alternate key
values completed.
-- File :NVE.USER99.IS_FILE : begin sorting the alternate key
values.
-- File :NVE.USER99.IS_FILE : sorting of the alternate key
values completed.
-- File :NVE.USER99.IS_FILE : begin building alternate key
indexes into the file.
-- File :NVE.USER99.IS_FILE : the ALTERNATE_KEY_1 index is
being built.
-- File :NVE.USER99.IS_FILE : AMP$APPLY_KEY_DEFINITIONS
completed building the alternate indexes into the file.
/
```

DELETE_KEY_DEFINITION CREAI Subcommand

Purpose Requests the deletion of an existing alternate key within a CREATE_ALTERNATE_INDEXES utility session.

Format

DELETE_KEY_DEFINITION or
DELKD
NAME=name
STATUS=status variable

Parameters

NAME or **N** or **KEY_NAME** or **KN**

Name of the alternate key to be deleted. This parameter is required.

Remarks

- The DELETE_KEY_DEFINITION subcommand requests deletion of an alternate key but does not actually delete the key from the file. The deletion remains pending until it is applied by either an APPLY_KEY_DEFINITIONS or QUIT subcommand or, it is canceled by a CANCEL_KEY_DEFINITIONS subcommand.

- For more information, see the NOS/VE Advanced File Management Usage manual.

Examples This CREATE_ALTERNATE_INDEXES session deletes an alternate key named ALTERNATE_KEY_1.

```
/create_alternate_indexes input=$user.is_file
creai/delete_key_definition key_name=alternate_key_1
creai/quit apply_key_definitions=yes
-- File :NVE.USER99.IS_FILE : begin deleting alternate key
definitions.
-- File :NVE.USER99.IS_FILE : Alternate key ALTERNATE_KEY_1
has been deleted.
-- File :NVE.USER99.IS_FILE : end deleting alternate key
definitions.
/
```

DISPLAY_KEY_DEFINITIONS CREAI Subcommand

Purpose Displays alternate-key definitions within a CREATE_ALTERNATE_INDEXES utility session.

Format

DISPLAY_KEY_DEFINITIONS or
DISPLAY_KEY_DEFINITION or
DISKD
NAMES=list of name
DISPLAY_OPTION=keyword
SAMPLE_RECORD_COUNT=integer or *keyword*
OUTPUT=file
STATUS=status variable

Parameters *NAMES* or *NAME* or *N* or *KEY_NAMES* or *KEY_NAME* or *KN*

Indicates the alternate key definitions displayed.

list of names

Displays the specified alternate-key definitions.

PENDING (P)

Displays only the pending alternate-key creations and deletions.

ALL (A)

Displays both pending and existing alternate-key definitions.

If the KEY_NAMES parameter is omitted, only the pending alternate-key creations and deletions are displayed.

DISPLAY_OPTION or *DO*

Indicates the contents of the display.

BRIEF (B)

Displays the key name, position, length, type, and state.

FULL (F)

Displays all information in the alternate-key definition.

SAMPLE_RECORDS (SR)

Displays only sample records with the alternate keys marked.

BRIEF_SAMPLE_RECORDS (BSR)

Displays the brief definition and the sample records.

FULL_SAMPLE_RECORDS (FSR)

Displays the full definition and the sample records.

ALL (A)

If the DISPLAY_OPTIONS parameter is omitted, the full definition and the sample records are displayed.

SAMPLE_RECORD_COUNT or *SRC*

Indicates the number of records displayed if the DISPLAY_OPTIONS parameter requests a sample record display.

integer

Displays the specified number of records. Values can be 0 through 4398046511103.

ALL (A)

Displays all records in the file.

The default is a one-record display.

OUTPUT or *O*

File to which the display is written. If the OUTPUT parameter is omitted, the display is written to file $OUTPUT.

Remarks

- A sample-record display shows the record contents in ASCII characters and hexadecimal digits with the alternate-key fields underscored. Each alternate key is shown separately by underscores as follows:

  If the concatenated-key or repeating-groups attributes are not defined for the key, the underscore characters indicate the alternate-key type (C for collated, I for integer, or U for uncollated).

  If the key is a concatenated key, the underscores for each key field include one or two letters indicating the order the fields are concatenated (a_, b_, and so forth up to z_ and then, aa, ba, ca, and so forth).

  If the alternate-key definition specifies repeating groups, the underscores for each alternate-key value in the record include a number (1, 2, and so forth).

- For more information, see the NOS/VE Advanced File Management Usage manual.

Examples

This CREATE_ALTERNATE_INDEXES session writes a display to file LIST. The listing includes all records in the file, marked with the proposed alternate-key ALTERNATE_KEY_2.

```
/create_alternate_indexes  input=$user.is_file
creai/crekd  key_name=alternate_key_2 ..
creai../key_position=0  key_length=2 ..
creai../repeating_group_length=20
creai/display_key_definitions  ..
creai../display_option=sample_records ..
creai../sample_record_count=all  output=list
creai/quit  apply_key_definitions=no
/
```

The following CREATE_ALTERNATE_INDEXES session contains a DISPLAY_KEY_DEFINITIONS subcommand for a default display, that is, a full definition of all pending alternate-key creations and deletions and a single sample record.

```
/create_alternate_indexes input=$user.is_file
creai/create_key_definition key_name=alternate_key_1 ..
creai../key_position=0 key_length=2 ..
creai/display_key_definitions
              Display_Key_Definitions     NOS/VE Keyed File Utilities 1.1
File = .NVE.USER99.IS_FILE

 KEY NAME                       POSITION LENGTH TYPE       STATE
 ------------------------------ -------- ------ ---------- ----------------
 ALTERNATE_KEY_1                       0      2 uncollated Creation pending
   Duplicate_Key_Values       : not_allowed
   Null_Suppression           : no
   Repeating_Groups_Specified
     Repeating_Group_Length   : 4
       Repeating_Group_Count  : repeat_to_end_of_record
================================================================================
RECORD 1 .......(in ascii) : This  is  the  first  record
                ( in hex ) : 546869732069732074686520666972737420726563 6F72
ALTERNATE_KEY_1            : 1_1_   2_2_   3_3_   4_4_   5_5_   6_6_
                (in ascii) : d .
                ( in hex ) : 642E
                           >
creai/quit apply_key_definitions=no
/
```

QUIT
CREAI Subcommand

Purpose Ends the CREATE_ALTERNATE_INDEXES utility session.

Format

QUIT or
QUI
APPLY_KEY_DEFINITIONS=boolean or *keyword*
ERROR_LIMIT=integer
STATUS=status variable

Parameters *APPLY_KEY_DEFINITIONS* or *APPLY_KEY_DEFINITION* or *AKD*

Indicates how pending alternate-key creation and deletion requests are processed.

APPLY (A), TRUE (ON or YES)

Apply all pending creation and deletion requests.

CANCEL (C), FALSE (OFF or NO)

Cancel all pending creation and deletion requests.

This parameter is required if creation or deletion requests are pending.

ERROR_LIMIT or *EL*

Number of trivial (nonfatal) errors allowed for the apply operation (integer from 0 through 4398046511103 [2**42-1]).

0 is the default value and indicates no limit.

See the APPLY_KEY_DEFINITIONS command description for a description of apply error processing.

Remarks

- The APPLY_KEY_DEFINITIONS parameter is required only if alternate-key creation or deletion requests are pending. In this case, you must specify whether to apply or cancel the pending requests.

  If you request application of the pending creations and deletions, the QUIT subcommand performs the same processing as the APPLY_KEY_DEFINITIONS subcommand before exiting the utility.

  If you request cancellation of the requests, the QUIT subcommand performs the same processing as the CANCEL_KEY_DEFINITIONS subcommand before exiting the utility.

- For more information, see the APPLY_KEY_DEFINITIONS and CANCEL_KEY_DEFINITIONS subcommand descriptions.

- For more information, see the NOS/VE Advanced File Management Usage manual.

Examples This CREATE_ALTERNATE_INDEXES session requests an alternate-key deletion and an alternate-key creation, but then cancels the requests.

```
/create_alternate_indexes file=$user.isfile
creai/delete_key_definition alternate_key_1
creai/create_key_definition alternate_key_1 ..
creai../key_position=0 key_length=5 key_type=integer
creai/quit apply_key_definitions=no
/
```

QUIT
CREKD Subcommand

Purpose Exits the CREATE_KEY_DEFINITION utility, ending concatenated-key specification.

Format **QUIT** or
QUI
STATUS=status variable

Remarks Entry of the QUIT subcommand returns you to the CREATE_ALTERNATE_INDEXES utility session. This is indicated by the prompt creai/.

Examples This CREATE_ALTERNATE_INDEXES session defines a concatenated alternate key having two pieces. The first piece is the ten bytes beginning at byte 5. (Remember, bytes are numbered from the left beginning with zero.) The second piece is the five-byte integer at the beginning of the record.

```
/create_alternate_indexes input=$user.is_file
creai/create_key_definition alternate_key_3 ..
creai../key_position=5 key_length=10 ..
creai../concatenated_pieces=yes
crekd/add_piece key_position=0 key_length=5 ..
crekd../key_type=integer
crekd/quit          "Exits CREATE_KEY_DEFINITIONS.
creai/quit no       "Exits CREATE_ALTERNATE_INDEXES without
/                   "applying the alternate-key definition.
```

SEPARATE_KEY_GROUPS
CREAI Subcommand

Remarks Reserved for site personnel, Control Data, or future use.

CREATE_INTERSTATE_CONNECTION 10

CREATE_INTERSTATE_CONNECTION Command

Purpose Establishes a NOS batch control point on a dual state system.

Format

CREATE_INTERSTATE_CONNECTION or
CREIC
PARTNER_JOB_CARD=string
STATUS=status variable

Parameters

PARTNER_JOB_CARD or *PJC*

Specifies the job statement parameters to be used for the NOS batch job. The parameter syntax must conform to NOS job statement rules.

Omission causes the NOS default job statement parameters to be used (an infinite time limit and no other parameters specified).

Remarks

- After you enter a CREATE_INTERSTATE_CONNECTION command, prompts are issued until you enter QUIT (QUI) or DELETE_INTERSTATE_CONNECTION (DELIC).
- While the interstate connection is open, you can enter any NOS/VE command (except another CREIC command). You can enter NOS commands to be executed on the NOS side of the dual state system through the EXECUTE_INTERSTATE_COMMAND command. The CREIC command is generally used in conjunction with the File Management Utility to migrate files between NOS and NOS/VE.
- For more information, see the NOS/VE Advanced File Management Usage manual.

Examples The following commands create an interstate connection, execute NOS commands (ATTACH, DEFINE, and COPY), and close the connection. FA is the CREATE_INTERSTATE_CONNECTION prompt for user input.

```
/create_interstate_connection partner_job_card=..
../'myjob,,64.'
FA/execute_interstate_command command='attach,oldfil.'
FA/execute_interstate_command command='define,newfil.'
FA/execute_interstate_command command=..
FA../'copy,oldfil,newfil.'
FA/delete_interstate_connection
/
```

DELETE_INTERSTATE_CONNECTION CREIC Subcommand

Purpose Ends a CREATE_INTERSTATE_CONNECTION session.

Format

DELETE_INTERSTATE_CONNECTION or
QUI or
QUIT or
DELIC

Parameters None.

Remarks For more information, see the Migration from NOS to NOS/VE manual.

EXECUTE_INTERSTATE_COMMAND CREIC Subcommand

Purpose Precedes all NOS commands when the interstate connection established by CREATE_INTERSTATE_CONNECTION (CREIC) is in effect.

Format

EXECUTE_INTERSTATE_COMMAND or
EXEIC
COMMANDS=list of string
STATUS=status variable

Parameters **COMMANDS** or **COMMAND** or **C**

A NOS command followed by a period. The command string can include up to 80 characters and must be enclosed in apostrophes. This command is required.

Remarks For more information, see the Migration From NOS to NOS/VE manual.

CREATE_OBJECT_LIBRARY 11

CREATE_OBJECT_LIBRARY Command

Purpose Begins a CREATE_OBJECT_LIBRARY utility session. The utility produces an object library or an object file and allows post-compilation manipulation of object or load modules. It can also produce a text version of certain kinds of modules on an object library.

Format

CREATE_OBJECT_LIBRARY or
CREOL
STATUS=status variable

Remarks

- The following files can be created by the GENERATE_LIBRARY subcommand of this utility. The utility issues a warning and does not process input files whose file attributes do not conform to the attributes listed in the right-hand column. The utility sets the accompanying file attributes listed for output files it creates. You can override attributes of a file with the SCL SET_FILE_ATTRIBUTE command.

File Created	Attributes Given to the File
Object library	FILE_CONTENT=OBJECT FILE_STRUCTURE=LIBRARY
Object file	FILE_CONTENT=OBJECT FILE_STRUCTURE=DATA
SCL procedure file	FILE_CONTENT=LEGIBLE FILE_PROCESSOR=SCL FILE_STRUCTURE=DATA
File containing the subcommands that define message modules	FILE_CONTENT=LEGIBLE FILE_PROCESSOR=SCL FILE_STRUCTURE=DATA

- The CREOL session ends when you enter the QUIT subcommand.
- For more information, see the NOS/VE Object Code Management manual.

Examples The following is a sequence that removes an object library from the command list, creates a new version of the object library from the modules on file $LOCAL.LGO, then adds the object library to the command list.

```
/delete_command_list_entry entry=$local.my_commands
/create_object_library
COL/add_module $local.lgo
COL/generate_library $local.my_commands
COL/quit
/create_command_list_entry entry=$local.my_commands
/
```

ADD_MODULE
CREOL Subcommand

Purpose Adds one or more modules to the module list.

Format

ADD_MODULE or
ADD_MODULES or
ADDM
LIBRARY=list of file
MODULE=list of range of any
PLACEMENT=keyword
DESTINATION=any
STATUS=status variable

Parameters

LIBRARY or **LIBRARIES** or **L**

Object files, SCL procedure files, or object library files containing the modules to be added. This parameter is required.

MODULE or *MODULES* or *M*

Modules to be added.

You use a string value for a module whose name is not an SCL name. Some examples of such module names are: a COBOL module, where a hyphen character (-) may be part of the name, and a C function, where lower case is significant.

If MODULE is omitted, all modules on the files specified on the LIBRARY parameter are added.

PLACEMENT or *P*

Indicates whether the added modules are placed before or after the module specified on the DESTINATION parameter. Options are:

BEFORE (B)

Modules added before the destination module.

AFTER (A)

Modules added after the destination module.

If PLACEMENT is omitted, AFTER is used.

DESTINATION or *D*

Module before or after which the added modules are placed.

If DESTINATION is omitted, the location depends on the PLACEMENT parameter value. If PLACEMENT=BEFORE, the modules are placed at the beginning of the module list; if PLACEMENT=AFTER, the modules are placed at the end of the module list.

Remarks

- The ADD_MODULE subcommand can specify object files, SCL procedure files, or object libraries. The CREOL utility adds modules from files in the order you specify the files on the LIBRARY parameter. If you do not want to use all modules in the files, specify the modules to be added on the MODULE parameter.

- Unless specified otherwise by the PLACEMENT and DESTINATION parameters, the subcommand adds each module to the end of the module list. If the module list already contains a module with the same name, a warning status message is returned and the module is not added.

- The ADD_MODULE subcommand does not replace modules in the module list. To replace modules, enter a REPLACE_MODULES subcommand. To add and replace modules, enter a COMBINE_MODULES subcommand.

- For more information, see the NOS/VE Object Code Management manual.

Examples The following subcommand adds all modules on files BINARY1 and BINARY2 to the beginning of the module list.

```
COL/add_module (binary1,binary2) placement=before
```

BIND_MODULE
CREOL Subcommand

Purpose Subcommand used in a restructuring procedure to bind component modules into a single load module. This subcommand is not recommended for your use. The subcommand description is provided only to help you interpret the commands in a restructuring procedure. To create a new module by binding component modules, you should use the subcommand CREATE_MODULE.

Format

BIND_MODULE or
BINM
MODE=keyword
NAME=name
FILE=file
STARTING_PROCEDURE=any
SECTION_ORDER=list of any
PRESET_VALUE=keyword
INCLUDE_BINARY_SECTION_MAPS=boolean
OUTPUT=file
STATUS=status variable

Parameters

MODE or M

Indicates whether additional BIND_MODULE subcommands for the module follow this subcommand. Options are:

CONTINUE

More BIND_MODULE subcommands follow.

QUIT

This is the last BIND_MODULE subcommand for the module.

This parameter is required.

NAME or *N*

Name of the new module. This parameter is required only on the first BIND_MODULE subcommand for the module.

You use a string value for a module whose name is not an SCL name.

FILE or *F*

File containing the modules to be bound. This parameter is required only on the first BIND_MODULE subcommand for the module.

STARTING_PROCEDURE or *SP*

Name of the transfer symbol for the new module.

You use a string value for a transfer symbol whose name is not an SCL name.

If STARTING_PROCEDURE is omitted, the last transfer symbol encountered is used.

SECTION_ORDER or *SO*

Code chapter ordering for the component modules in the new module. Each item in the list contains a module name and its chapter ordinal.

PRESET_VALUE or *PV*

Specifies text record reduction as follows.

ZERO (Z)

Reduces the number of individual text records in an object module. Reducing the number of records reduces the amount of time it takes to load the module.

If PRESET_VALUE is omitted, the number of text records is not reduced.

INCLUDE_BINARY_SECTION_MAPS or *IBSM*

Indicates whether the binary section map is included in the information element for the bound module.

OUTPUT or *O*

File to which the section map for the new module is written. This file can be positioned. If OUTPUT is omitted, no section map is written.

Remarks

- The new module is not generated until you enter a GENERATE_LIBRARY subcommand. Therefore, the section map for the module is not written on the file specified on the OUTPUT parameter until the module is generated.

- A restructuring procedure uses a sequence of BIND_MODULE subcommands to direct the generation of the load module. The first subcommand in the sequence must specify the module name and the file containing the modules to be bound. Each subcommand except the last in the sequence for the module must specify MODE=CONTINUE. The last subcommand in the sequence must specify MODE=QUIT. Refer to the Application Efficiency chapter of the Object Code Management manual for more information on restructuring.

- For more information, see the NOS/VE Object Code Management manual.

Examples

The following is a restructuring procedure generated for two object modules named EXAMP and NAND on file BIN3.

```
PROC MY_PROC(
  target_text,tt:file=:$LOCAL.BIN3
  library,l:file=:$LOCAL.MY_FILE
  module_name,mn:name=MY_FILE
  status)
    create_object_library
      bind_module name=$VALUE(module_name) ..
          file=$VALUE(target_text) mode=continue
      bind_module mode=continue section_order=((EXAMP   0))
      bind_module mode=continue section_order=((NAND    0))
      bind_module mode=quit
      generate_library library=$VALUE(library)
    quit
PROCEND
```

CHANGE_MODULE_ATTRIBUTE CREOL Subcommand

Purpose Changes one or more attributes of a module in the module list.

Format

CHANGE_MODULE_ATTRIBUTE or
CHANGE_MODULE_ATTRIBUTES or
CHAMA
MODULE=list of range of any
NEW_NAME=any
SUBSTITUTE=list of any
OMIT=list of any
GATE=list of any
NOT_GATE=list of any
STARTING_PROCEDURE=any
OMIT_LIBRARY=list of name
ADD_LIBRARY=list of name
RETAIN=list of any
NOT_RETAIN=list of any
OMIT_NON_RETAINED_ENTRY_POINTS=boolean
OMIT_DEBUG_TABLES=list of keyword
COMMENT=string
APPLICATION_IDENTIFIER=name or *keyword*
STATUS=status variable

Parameters

MODULE or **M**

Module whose attributes are changed.

You use a string value for an entry point whose name is not an SCL name. Some examples of such module names are: a COBOL module, where a hyphen character (-) may be part of the name, and a C function, where lower case is significant.

ALL may be specified to change the attributes of all modules.

This parameter is required.

NEW_NAME or *NN*

New module name.

You use a string value for an entry point whose name is not an SCL name. Examples of such names are the same as for the MODULE parameter.

If ALL is specified in the MODULE parameter, NEW_NAME should not be used.

If NEW_NAME is omitted, the module name is not changed.

SUBSTITUTE or *SUBSTITUTES* or *S*

List of name substitutions. Each item in the list specifies two names: the name to be replaced and the name to replace it.

You use a string value for an entry point whose name is not an SCL name. Examples of such names are the same as for the MODULE parameter.

The name to be replaced can be an entry point name or the name of a CYBIL variable with the XDCL attribute. If SUBSTITUTE is omitted, no names are changed.

OMIT or *O*

List of names whose definitions are removed from the module. The name to be removed can be an entry point name or the name of a CYBIL variable with the XDCL attribute.

You use a string value for an entry point whose name is not an SCL name. If OMIT is omitted, no name definitions are removed.

GATE or *GATES* or *G*

List of entry points to which the gate attribute is added.

You use a string value for an entry point whose name is not an SCL name. An example of such an entry point is in the COBOL language, where a hyphen character (-) may be part of the name.

If ALL is specified, the gate attribute is added to all entry points in the module.

If GATE is omitted, the gate attribute is not added to any entry point name.

NOT_GATE or *NOT_GATES* or *NG*

List of entry points from which the gate attribute is removed.

You use a string value for an entry point whose name is not an SCL name. An example of such an entry point is in the COBOL language, where a hyphen character (-) may be part of the name.

If ALL is specified, the gate attribute is removed from all entry points in the module.

If NOT_GATE is omitted, the gate attribute is not removed from any entry point.

STARTING_PROCEDURE or *SP*

Name of the entry point where execution begins.

You use a string value for an entry point whose name is not an SCL name. An example of such an entry point is in the COBOL language, where a hyphen character (-) may be part of the name.

If STARTING_PROCEDURE is omitted, the starting procedure is not changed.

OMIT_LIBRARY or *OMIT_LIBRARIES* or *OL*

List of local file names to be removed from the object text (text-embedded libraries). The local file names specify object libraries to be added to the program library list when the module is loaded. All specifications for these files are removed from the object text when the load module is written on the new object library.

If OMIT_LIBRARY is omitted, no library specifications are removed.

ADD_LIBRARY or *ADD_LIBRARIES* or *AL*

List of local file names to be added to the object text (text-embedded libraries). The local file names specify object libraries to be added to the program library list when the module is loaded. The CREOL utility adds the file specifications to each module when it writes the load module on the new object library.

If ADD_LIBRARY is omitted, no library specifications are added.

RETAIN or *R*

List of additional entry points to be given the retain attribute. An entry point with the retain attribute is kept in a new module created by combining this module with other modules.

You use a string value for an entry point whose name is not an SCL name. An example of such an entry point is in the COBOL language, where a hyphen character (-) may be part of the name.

If ALL is specified, the retain attribute is given to all entry points.

If RETAIN is omitted, no additional entry points are given the retain attribute.

NOT_RETAIN or *NR*

List of entry points from which the retain attribute is removed. Without the retain attribute, the entry point is removed from any new module created by combining this module with other modules that reference the entry point.

You use a string value for an entry point whose name is not an SCL name. An example of such an entry point is in the COBOL language, where a hyphen character (-) may be part of the name.

If ALL is specified, the retain attribute is removed from all entry points.

If NOT_RETAIN is omitted, the retain attribute is not removed from any entry point.

OMIT_NON_RETAINED_ENTRY_POINTS or *ONREP*

Specifies that all entry points are removed from the module unless they are explicitly retained. If OMIT_NON_RETAINED_ENTRY_POINTS is omitted, all entry points are retained.

OMIT_DEBUG_TABLES or *OMIT_DEBUG_TABLE* or *ODT*

List of one or more keywords indicating the debug tables to be omitted when the module is loaded. Options are:

LINE_TABLE (LT)

Omits the debug table containing line numbers that correspond to the module.

SYMBOL_TABLE (ST)

Omits the debug table containing the names and addresses of the program variables in the module.

PARAMETER_CHECKING (PC)

Omits parameter checking records in the module.

ALL

Omits all the debug tables.

Using the OMIT_DEBUG_TABLE parameter causes the module to load faster. If it is omitted, any debug tables in the module are included when the module is loaded. (Debug tables are generated during compilation, if requested by the compiler command.)

COMMENT or *C*

Commentary stored in the module header (from one through 40 characters). If COMMENT is omitted, the commentary is not changed.

APPLICATION_IDENTIFIER or *AI*

Name of application identifier stored in the module header and included on the application accounting statistics when the software is executed. Application identifiers can be specified only for program description, command procedure, and load modules. Only a user with the SYSTEM_ADMINISTRATION or APPLICATION_ADMINISTRATION capability can specify an application identifier.

If the keyword $UNSPECIFIED is used, the application identifier is removed.

If an application identifier is already assigned and if the APPLICATION_IDENTIFIER parameter is omitted, the application identifier is not changed.

Remarks

- The MODULE parameter specifies the module whose attributes are changed. The module must be in the current module list.
- You specify an attribute parameter for each attribute to be changed. If you omit an attribute parameter, the attribute value is not changed.

- The CHANGE_MODULE_ATTRIBUTES subcommand only changes the attributes of the module written by a subsequent GENERATE_LIBRARY subcommand. It does not change the attributes of the original module.
- For more information, see the NOS/VE Object Code Management manual.

Examples The following subcommand changes the name of entry point EXAMPLE in module MY_MODULE to EXAMPLE_1.

```
COL/change_module_attributes my_module ..
COL../substitute=((example,example_1))
```

CHANGE_PROGRAM_DESCRIPTION CREOL Subcommand

Purpose Changes the components of a program description

Format **CHANGE_PROGRAM_DESCRIPTION** or **CHAPD**

NAME=any
FILE=list of any or *keyword*
LIBRARY=list of any or *keyword*
MODULE=list of any or *keyword*
STARTING_PROCEDURE=any or *keyword*
LOAD_MAP=any or *keyword*
LOAD_MAP_OPTION=list of keyword
TERMINATION_ERROR_LEVEL=keyword
PRESET_VALUE=keyword
STACK_SIZE=integer or *keyword*
ABORT_FILE=any or *keyword*
DEBUG_INPUT=any or *keyword*
DEBUG_OUTPUT=any or *keyword*
DEBUG_MODE=boolean or *keyword*
AVAILABILITY=keyword
SCOPE=keyword
LOG_OPTION=keyword
APPLICATION_IDENTIFIER=name or *keyword*
ARITHMETIC_OVERFLOW=boolean or *keyword*
ARITHMETIC_LOSS_OF_SIGNIFICANCE=boolean or *keyword*
DIVIDE_FAULT=boolean or *keyword*
EXPONENT_OVERFLOW=boolean or *keyword*

EXPONENT_UNDERFLOW=boolean or *keyword*
FP_INDEFINITE=boolean or *keyword*
FP_LOSS_OF_SIGNIFICANCE=boolean or *keyword*
INVALID_BDP_DATA=boolean or *keyword*
STATUS=status variable

Parameters

NAME or **N**

Specifies the name of the program being changed. This parameter is required.

FILE or *FILES* or *F*

List of object files or object libraries to be unconditionally loaded when the program is executed.

Path values containing $FAMILY, $USER, or $SYSTEM elements can be supplied as strings to be evaluated when the program description is used.

If the FILE parameter is omitted, the FILE parameter of the program description is not changed. If $UNSPECIFIED is used, the FILE parameter is removed from the program description.

LIBRARY or *LIBRARIES* or *L*

List of library files to be added to the program library list when the program is executed. A file value is evaluated when the object library is generated. Path values containing $FAMILY, $USER, or $SYSTEM elements can be supplied as strings to be evaluated when the program description is used.

If $UNSPECIFIED is used, the LIBRARY parameter is removed from the program description.

The keyword OSF$TASK_SERVICES_LIBRARY specifies the system table, and keyword OSF$CURRENT_LIBRARY represents the library that contains the program description being changed.

If the LIBRARY parameter is omitted, the LIBRARY parameter of the program description is not changed.

MODULE or *MODULES* or *M*

List of modules to be loaded from the program library list when the program is executed. The modules are loaded in the order in which they are specified in the list.

You use a string value for a module whose name is not an SCL name. Some examples of such module names are: a COBOL module, where a hyphen character (-) may be part of name, and a C function, where lower case is significant.

If the MODULE parameter is omitted, the MODULE parameter of the program description is not changed. If $UNSPECIFIED is used, the MODULE parameter is removed from the program description.

STARTING_PROCEDURE or *SP*

Name of the entry point at which program execution begins.

You use a string value for an entry point whose name is not an SCL name. An example of such an entry point is in the COBOL language, where a hyphen character (-) may be part of the name.

If the STARTING_PROCEDURE parameter is omitted, the STARTING_PROCEDURE parameter of the program description is not changed. If $UNSPECIFIED is used, the STARTING_PROCEDURE parameter is removed from the program description.

LOAD_MAP or *LM*

File on which the load map is written. A file value is evaluated when the object library is generated.

Path values containing $FAMILY, $USER, or $SYSTEM elements can be supplied as strings to be evaluated when the program description is used.

If the LOAD_MAP parameter is omitted, the LOAD_MAP parameter of the program description is not changed. If $UNSPECIFIED is used, the LOAD_MAP parameter is removed from the program description.

LOAD_MAP_OPTION or *LOAD_MAP_OPTIONS* or *LMO*

List one or more keywords indicating the information to include in the load map. Options are:

NONE

No load map is written.

SEGMENT (S)

Segment map.

BLOCK (B)

Block map.

ENTRY_POINT (EP)

Entry point map.

CROSS_REFERENCE (CR)

Entry point cross-reference.

ALL

Selects SEGMENT, BLOCK, ENTRY_POINT, and CROSS_REFERENCE.

$UNSPECIFIED

The LOAD_MAP_OPTION parameter is removed from the program description.

If the LOAD_MAP_OPTION parameter is omitted, the LOAD_MAP_OPTION parameter of the program description is not changed.

TERMINATION_ERROR_LEVEL or *TEL*

Specifies the severity level of error that terminates program loading. Options are:

WARNING (W)

Warning, error, or fatal severity level errors.

ERROR (E)

Error or fatal severity level errors.

FATAL (F)

Fatal severity level errors.

$UNSPECIFIED

The TERMINATION_ERROR_LEVEL parameter is removed from the program description.

If the TERMINATION_ERROR_LEVEL parameter is omitted, the TERMINATION_ERROR_LEVEL parameter of the program description is not changed.

PRESET_VALUE or *PV*

Value to store in all uninitialized data words. Options are:

ZERO (Z)

All zeroes.

FLOATING_POINT_INDEFINITE (FPI)

Floating-point indefinite value.

INFINITY (I)

Floating-point infinite value.

ALTERNATE_ONES (AO)

Alternating 0 and 1 bits; the leftmost (highest order) bit is 1.

UNSPECIFIED

The PRESET_VALUE parameter is removed from the program description.

If the PRESET_VALUE parameter is omitted, the parameter of the program description is not changed.

STACK_SIZE or *SS*

Maximum number of bytes in the run-time stack. The program uses the run-time stack for procedure call linkages and local variables. If STACK_SIZE is omitted, the system default value is used. You can display the default stack size by entering a DISPLAY_PROGRAM_ATTRIBUTE command. If $UNSPECIFIED is used, the STACK_SIZE parameter is removed from the program description.

ABORT_FILE or *AF*

File containing Debug commands to be processed if the program aborts. The commands are executed only if the program is not executed in Debug mode. A file value is evaluated when the object library is generated.

Path values containing $FAMILY, $USER, or $SYSTEM elements can be supplied as strings to be evaluated when the program description is used.

If ABORT_FILE is omitted, the program description for the ABORT_FILE parameter is not changed. If $UNSPECIFIED is used, the ABORT_FILE parameter is removed from the program description.

DEBUG_INPUT or *DI*

File containing Debug commands. The commands are read only if the program is executed under the control of Debug (refer to the DEBUG_MODE parameter). This file can be positioned. A file value is evaluated when the object library is generated.

Path values containing $FAMILY, $USER, or $SYSTEM elements can be supplied as strings to be evaluated when the program description is used.

If DEBUG_INPUT is omitted, the DEBUG_INPUT parameter of the program description is not changed. If $UNSPECIFIED is used, the DEBUG_INPUT parameter is removed from the program description.

DEBUG_OUTPUT or *DO*

File on which Debug output is written. Output is written only if the program is executed in Debug mode. This file can be positioned. A file value is evaluated when the object library is generated.

Path values containing $FAMILY, $USER, or $SYSTEM elements can be supplied as strings to be evaluated when the program description is used.

If DEBUG_OUTPUT is omitted, the DEBUG_OUTPUT parameter of the program description is not changed. If $UNSPECIFIED is used, the DEBUG_OUTPUT parameter is removed from the program description.

DEBUG_MODE or *DM*

Indicates whether the program is to be run under the control of Debug. (For information on using Debug, refer to the program's specific source language manual.) Options are:

ON

Program executed under control of the Debug program.

OFF

Program executed without the Debug program.

If the DEBUG_MODE parameter is omitted, the DEBUG_MODE parameter of the program description is not changed. If $UNSPECIFIED is used, the DEBUG_MODE parameter is removed from the program description.

AVAILABILITY or *A*

Specifies whether or not the program description is made known to users as a command. Options are:

ADVERTISED (A)

Program description appears in the output produced by the DISPLAY_COMMAND_LIST_ENTRY command (and in similar situations).

HIDDEN (H)

Program description is suppressed from the output produced by DISPLAY_COMMAND_LIST_ENTRY command (and in similar situations).

If this parameter is omitted, the AVAILABILITY parameter of the program description is not changed.

SCOPE or *S*

Reserved for future use.

LOG_OPTION or *LO*

Reserved for future use.

APPLICATION_IDENTIFIER or *AI*

Name of application identifier stored in the module header and included on the application accounting statistics when the software is executed. Only a user with the SYSTEM_ADMINISTRATION or APPLICATION_ADMINISTRATION capability can specify an application identifier.

If the keyword $UNSPECIFIED is used, the application identifier is removed. If the APPLICATION_IDENTIFIER parameter is omitted, the application identifier is not changed.

ARITHMETIC_OVERFLOW or *AO*

This parameter specifies whether or not the hardware condition ARITHMETIC_OVERFLOW causes an interrupt. Valid specifications are:

ON

ARITHMETIC_OVERFLOW is enabled. The condition causes an interrupt.

OFF

ARITHMETIC_OVERFLOW is disabled. The condition does not cause an interrupt.

ARITHMETIC_LOSS_OF_SIGNIFICANCE or *ALOS*

This parameter specifies whether or not the hardware condition ARITHMETIC_LOSS_OF_SIGNIFICANCE causes an interrupt. Valid specifications are:

ON

ARITHMETIC_LOSS_OF_SIGNIFICANCE is enabled. The condition causes an interrupt.

OFF

ARITHMETIC_LOSS_OF_SIGNIFICANCE is disabled. The condition does not cause an interrupt.

DIVIDE_FAULT or *DF*

This parameter specifies whether or not the hardware condition DIVIDE_FAULT causes an interrupt. Valid specifications are:

ON

DIVIDE_FAULT is enabled. The condition causes an interrupt.

OFF

DIVIDE_FAULT is disabled. The condition does not cause an interrupt.

EXPONENT_OVERFLOW or *EO*

This parameter specifies whether or not the hardware condition EXPONENT_OVERFLOW causes an interrupt. Valid specifications are:

ON

EXPONENT_OVERFLOW is enabled. The condition causes an interrupt.

OFF

EXPONENT_OVERFLOW is disabled. The condition does not cause an interrupt.

EXPONENT_UNDERFLOW or *EU*

This parameter specifies whether or not the hardware condition EXPONENT_UNDERFLOW causes an interrupt. Valid specifications are:

ON

EXPONENT_UNDERFLOW is enabled. The condition causes an interrupt.

OFF

EXPONENT_UNDERFLOW is disabled. The condition does not cause an interrupt.

FP_INDEFINITE or *FPI* or *FI*

This parameter specifies whether or not the hardware condition FP_INDEFINITE causes an interrupt. Valid specifications are:

ON

FP_INDEFINITE is enabled. The condition causes an interrupt.

OFF

FP_INDEFINITE is disabled. The condition does not cause an interrupt.

FP_LOSS_OF_SIGNIFICANCE or *FPLOS* or *FLOS*

This parameter specifies whether or not the hardware condition FP_LOSS_OF_SIGNIFICANCE causes an interrupt. Valid specifications are:

ON

FP_LOSS_OF_SIGNIFICANCE is enabled. The condition causes an interrupt.

OFF

FP_LOSS_OF_SIGNIFICANCE is disabled. The condition does not cause an interrupt.

INVALID_BDP_DATA or *IBDPD* or *IBD*

This parameter specifies whether or not the hardware condition INVALID_BDP_DATA causes an interrupt. Valid specifications are:

ON

INVALID_BDP_DATA is enabled. The condition causes an interrupt.

OFF

INVALID_BDP_DATA is disabled. The condition does not cause an interrupt.

Remarks

- To allow users the option of rescinding a previously specified value or not including a given parameter in the CHAPD command, the keyword $UNSPECIFIED may be used for some parameters. This removes the parameter from the description. The result of using the $UNSPECIFIED is the same as not supplying the parameter on the CREATE_PROGRAM_DESCRIPTION subcommand. When the program is executed, the corresponding job default program attribute value is used.
- For more information, see the NOS/VE Object Code Management manual.

Examples See the NOS/VE Object Code Management manual for a detailed example.

COMBINE_MODULE CREOL Subcommand

Purpose Adds new modules and replaces existing modules in the module list.

Format

COMBINE_MODULE or
COMBINE_MODULES or
COMM
LIBRARY=list of file
MODULE=list of range of any
PLACEMENT=keyword
DESTINATION=any
STATUS=status variable

Parameters

LIBRARY or **LIBRARIES** or **L**

Object files, SCL procedure files, or object library files containing the modules to be combined. This parameter is required.

MODULE or *MODULES* or *M*

Modules to be combined.

You use a string value for a module whose name is not an SCL name. Some examples of such module names are: a COBOL module, where a hyphen character (-) may be part of the name, and a C function, where lowercase is significant.

If MODULE is omitted, all modules on the specified files or libraries are combined.

PLACEMENT or *P*

Indicates whether the added modules are placed before or after the module specified on the DESTINATION parameter. Options are:

BEFORE (B)

Modules added before the destination module.

AFTER (A)

Modules added after the destination module.

If PLACEMENT is omitted, AFTER is used.

DESTINATION or *D*

Module before or after which the added modules are placed.

This parameter does not affect the location of replacement modules. A replacement module is always placed in the same location as the module it replaces.

If DESTINATION is omitted, added modules are placed according to the PLACEMENT parameter value. If PLACEMENT=BEFORE, the modules are placed at the beginning of the library. If PLACEMENT=AFTER, the modules are placed at the end of the library.

Remarks

- The COMBINE_MODULES subcommand can specify object files, SCL procedure files, or object libraries that are processed in the order you specify the files on the LIBRARY parameter.
- The COMBINE_MODULES subcommand checks for duplicate modules in the specified files and reports an error if duplicates are found.

  You can, however, combine modules in libraries with duplicate modules. You add one of the libraries to the module list with an ADD_MODULES subcommand and then perform a COMBINE_MODULES of the second library.
- If you do not want to use all modules in a file, specify the modules to be used on the MODULE parameter.
- A module to be combined replaces any module already existing with the same name in the module list. If the name is not already in the module list, the module to be combined is added to the module list.
- A replacement module is placed in the module list at the same location as the module it replaces. An added module is added at the end of the list unless you specify another location with the DESTINATION and PLACEMENT parameters. You can change the module order later with a REORDER_MODULES subcommand.
- For more information, see the NOS/VE Object Code Management manual.

Examples

The following subcommand combines all modules in files MY_LIBRARY and YOUR_LIBRARY with the modules already in the module list.

```
COL/combine_module (my_library,your_library)
```

CREATE_APPLICATION_MENU CREMM Subcommand

Purpose Initiates the CREATE_APPLICATION_MENU utility session.

Format

CREATE_APPLICATION_MENU or
CREAM
NAME=name
STATUS=status variable

Parameters **NAME** or **N**

Specifies the name of the application menu. The NAME parameter is a string containing 1 through 31 characters. This parameter is required.

Remarks For more information, see the NOS/VE Object Code Management manual.

CREATE_BRIEF_HELP_MESSAGE CREMM Subcommand

Purpose Creates a brief description of a command. The complete description is generated by the CREATE_FULL_HELP_MESSAGE subcommand.

Format

CREATE_BRIEF_HELP_MESSAGE or
CREBHM
COLLECT_TEMPLATE_UNTIL=string
STATUS=status variable

Parameters *COLLECT_TEMPLATE_UNTIL* or *CTU*

Specifies the termination string used in collecting the template of the brief help message. If the COLLECT_TEMPLATE_UNTIL parameter is omitted, the string '**' is assumed.

Remarks For more information, see the NOS/VE Object Code Management manual.

Examples The following creates a brief help message.

```
CMM/create_brief_help_message
? This is a little bit of help.
? **
CMM/
```

CREATE_FULL_HELP_MESSAGE CREMM Subcommand

Purpose Creates a message containing a complete description of the command. A brief description is generated with the CREATE_BRIEF_HELP_MESSAGE subcommand.

Format

CREATE_FULL_HELP_MESSAGE or
CREFHM
COLLECT_TEMPLATE_UNTIL=string
STATUS=status variable

Parameters *COLLECT_TEMPLATE_UNTIL* or *CTU*

Specifies the termination string used in collection of the template of the full help message. If the COLLECT_TEMPLATE_UNTIL parameter is omitted, the string '**' is assumed.

Remarks For more information, see the NOS/VE Object Code Management manual.

Examples The following creates a full help message.

```
CMM/create_full_help_message
? This is a complete description, providing more
? detailed instructions than the brief help message.
? **
CMM/
```

CREATE_LINKED_MODULE CREOL Subcommand

Purpose Creates a new prelinked module from an existing module and adds it to the module list.

Format

CREATE_LINKED_MODULE or
CRELM
NAME=name
COMPONENT=list of any
RING_BRACKETS=list of integer
RETAIN_COMMON_BLOCK=list of any
IGNORE_SECTION_NAMES=boolean
STARTING_SEGMENT=integer

OUTPUT=file
DEBUG_TABLE=file
NEXT_AVAILABLE_SEGMENT=integer variable
APPLICATION_IDENTIFIER=name
STATUS=status variable

Parameters

NAME or N

Name of the new prelinked module.

You use a string value for a module whose name is not an SCL name. Some examples of such module names are: a COBOL module, where a hyphen character (-) may be part of the name, and a C function, where lower case is significant.

This parameter is required.

COMPONENT or COMPONENTS or C

Component modules of the new module. Each item in the list is a list consisting of a file name followed by a series of module names on the file, which are to be used. A range of names may be specified. If no module names are specified for a file, all modules on the file are used.

You use a string value for a module whose name is not an SCL name. Some examples of such module names are: a COBOL module, where a hyphen character (-) may be part of the name, and a C function, where lower case is significant.

NOTE

A component module can be only an object, load, or bound module.

This parameter is required. At least one file must be specified.

RING_BRACKETS or RB

Specifies three integers representing the r1, r2, and r3 ring execution values for the new module. The ring values can be from 3 through 15. This parameter is required.

RETAIN_COMMON_BLOCK or *RETAIN_COMMON_BLOCKS* or *RCB*

Specifies which common block names are retained in the new modules. The keyword ALL specifies that all common blocks are retained.

You use a string value for a common block whose name is not an SCL name.

If RETAIN_COMMON_BLOCK is omitted, no common blocks are retained.

IGNORE_SECTION_NAMES or *ISN*

Specifies whether working storage sections with different names should be placed in unique segments. If IGNORE_SECTION_NAMES is omitted or IGNORE_SECTION_NAMES=TRUE, sections with similar access attributes (read and write) are placed in the same segments, regardless of the section names.

STARTING_SEGMENT or *SS*

First segment number to use in prelinking this module. The STARTING_SEGMENT parameter provides a unique starting segment number. It is used only when creating multiple prelinked modules that are loaded together.

Use the NEXT_AVAILABLE_SEGMENT parameter to generate the integer value for the STARTING_SEGMENT parameter on the next CREATE_LINKED_MODULE subcommand.

Integer values are 0 through 4,095. The operating system reserves segments 36 through 63 for prelinked programs. Each program must fit into these segments. Do not use segments 0 through 35, and 64 through 4,095.

If STARTING_SEGMENT is omitted, the integer value 36 is used as the starting segment number.

OUTPUT or *O*

File to which the prelink information and diagnostics are written. This file can be positioned.

If OUTPUT is omitted, information is written to file $LOCAL.LINKMAP.

DEBUG_TABLE or *DT*

File to which the table containing binary debug information is written. This parameter is for Control Data internal use only.

If DEBUG_TABLE is omitted, no debug information is written.

NEXT_AVAILABLE_SEGMENT or *NAS*

Integer variable to contain the next available segment number. Use this parameter only when you are creating multiple prelinked modules that are to be loaded together. This parameter generates unique segment numbers to be used by the STARTING_SEGMENT parameter on the next CREATE_LINKED_MODULE subcommand.

If NEXT_AVAILABLE_SEGMENT is omitted, no segment number is returned.

APPLICATION_IDENTIFIER or *AI*

Name of the application identifier stored in the module header and included on the application accounting statistics when the software is executed. Only a user with the SYSTEM_ADMINISTRATION or APPLICATION_ADMINISTRATION capability can specify an application identifier.

If an application identifier is placed on a load module, the module is assumed to be a unit-measured application.

If APPLICATION_IDENTIFIER is omitted, no application identifier is assigned to the module.

Remarks

- When building programs that consist of multiple prelinked modules, all predefined segment numbers must be unique for the entire load sequence.
- Use the STARTING_SEGMENT parameter on the CREATE_LINKED_MODULE subcommand to specify the first reserved segment number for a module. This allows modules that are prelinked separately to be used together at execution time.
- The system issues a warning diagnostic message for all text-embedded libraries encountered during prelinking. If the warning is ignored, the loader attempts to satisfy text-embedded library references at load time.

- During prelinking, an output file is generated that contains diagnostics and information on how the program was prelinked. This link map's default file name is $LOCAL.LINKMAP.
- Do not prelink COBOL programs that use CALL and CANCEL into a single module because CALL will try to overlay a single component module that is no longer available.
- Once you have prelinked modules, they can no longer be debugged using the interactive debugger. The debug information written to the file specified by the DEBUG_TABLE parameter is not the same as the debug tables used by the interactive debugger.
- For more information, see the NOS/VE Object Code Management manual.

Examples The following sequence creates a prelinked module named PRELINKED_MODULE from component BOUND_PRODUCT with ring brackets of (11,11,11). The module is then put in object library PRELINKED_PRODUCT and executed.

```
/create_object_library
COL/create_linked_module name=prelinked_module ..
COL../component=bound_product ..
COL../ring_brackets=(11,11,11)
COL/generate_library prelinked_product
COL/quit
/execute_task ..
../starting_procedure=product_entry_point ..
../library=prelinked_product
```

CREATE_MENU_CLASS CREAM Subcommand

Purpose Creates a class for an application menu. A class is defined as a name for a submenu. This subcommand allows you to identify a grouping of menu items. Up to 16 menu classes can be defined for a menu.

Format

CREATE_MENU_CLASS or
CREMC
NAME=string
STATUS=status variable

Parameters

NAME or **N**

Specifies the identification for the menu class being defined. Menu class names must be unique within a menu. The NAME parameter is a string containing 1 through 31 characters. This parameter is required.

Remarks

For more information, see the NOS/VE Object Code Management manual.

CREATE_MENU_ITEM CREAM Subcommand

Purpose

Creates an item for an application menu. A menu represents a particular action to be performed by the application program, or a particular option for such an action. Up to 20 menu items can be defined for each menu class.

Format

CREATE_MENU_ITEM or
CREMI
KEY=keyword
SHIFT=boolean
CLASS=string
SHORT_LABEL=string
ALTERNATE_SHORT_LABEL=string
LONG_LABEL=string
ALTERNATE_LONG_LABEL=string
PAIR_WITH_PREVIOUS=boolean
STATUS=status variable

Parameters

KEY or *K*

Specifies the key on a terminal keyboard that is associated with the menu item. The name of the key and the associated SHIFT parameter must be unique within the menu. Selectable keys are f1, f2, f3, f4, f5, f6, f7, f8, f9, f10, f11, f12, f13, f14, f15, f16, next, help, stop, back, up, down, forward, backward, edit, data, insert_line, delete_line, home, clear, clear_eol_menu_item, delete_char_menu_item, insert_char_menu_item, and undo.

Omission of the KEY parameter causes no assignment of the menu item to a key. The menu item is automatically assigned to a key, however, when the menu is used.

SHIFT

Indicates whether the menu item is associated with a shifted key (YES) or an unshifted key (NO).

Omission of the SHIFT parameter assumes an unshifted key. If the KEY parameter is omitted, the SHIFT parameter is ignored.

CLASS or *C*

Specifies the menu class for this menu item. The CLASS parameter is a string containing 1 through 31 characters.

Omission of the CLASS parameter causes the most recently created menu class to be used. If no menu classes have been defined, an error results.

SHORT_LABEL or **SL**

Provides a short label to represent this menu item for the application user. The SHORT_LABEL parameter is a string containing 1 through 6 characters. this parameter is required.

ALTERNATE_SHORT_LABEL or *ASL*

Provides a short label when the meaning of the menu item is toggled. The ALTERNATE_SHORT_LABEL parameter is a string containing 1 through 6 characters.

Omission of the ALTERNATE_SHORT_LABEL parameter causes the value for the SHORT_LABEL parameter to be used; the menu item's meaning does not toggle.

LONG_LABEL or *LL*

Provides a long label to represent this menu item for the application user. The LONG_LABEL parameter is a string containing 1 through 31 characters.

Omission of the LONG_LABEL parameter causes the SHORT_LABEL parameter to be used.

ALTERNATE_LONG_LABEL or *ALL*

Provides a long label when the meaning of the menu item is toggled. The ALTERNATE_LONG_LABEL parameter is a string containing 1 through 31 characters.

Omission of the ALTERNATE_LONG_LABEL parameter causes the value for the LONG_LABEL parameter to be used; the menu item's meaning does not toggle.

PAIR_WITH_PREVIOUS or *PWP*

Indicates (YES) that this menu item is to be paired with the most recently created menu item during automatic assignment of menu items to keys.

Omission causes NO to be assumed; that is, there is no assignment preference in pairing this menu item with other menu items.

Remarks For more information, see the NOS/VE Object Code Management manual.

CREATE_MESSAGE_MODULE CREOL Subcommand

Purpose Initiates the construction of a message module. It also initiates the CREATE_MESSAGE_MODULE utility.

Format

CREATE_MESSAGE_MODULE or
CREMM
NAME=any
MANUAL=any
NATURAL_LANGUAGE=keyword
MERGE_OPTION=keyword
STATUS=status variable

Parameters **NAME** or **N**

Specifies the name of the message module being created. This parameter is required.

For status messages, any name can be specified.

For help and prompt messages, the name references the procedure or command for which the message module is being created. It must be in the form:

name$language

name is the seed name of the message module specified on either the parameter descriptor table (PDT) or the procedure header. *language* is the natural language used to compose the messages in in this module (it should be the same as that specified by the NATURAL_LANGUAGE parameter).

You use a string value for a message module whose name is not an SCL name.

MANUAL or *M*

Reserved for future use.

NATURAL_LANGUAGE or *NL*

Specifies the natural language used to compose the messages for the message module. Options are:

DANISH
DUTCH
ENGLISH
FINNISH
FLEMISH
FRENCH
GERMAN
ITALIAN
NORWEGIAN
PORTUGUESE
SPANISH
SWEDISH
US_ENGLISH

Omission causes US_ENGLISH to be used.

MERGE_OPTION or *MO*

Specifies whether the message module should be added, replaced, or combined with the new object library. Options are:

ADD (A)

Message module is added to the new library.

REPLACE (R)

Message module replaces an existing module on the library.

COMBINE (C)

Message module is placed in the new library whether a module with the same name is present or not. If one is present, it is replaced with the new module.

Omission of the MERGE_OPTION parameter causes COMBINE to be used.

Remarks

- CYBIL programmers can use GENERATE_MESSAGE_TEMPLATE to facilitate creating message modules for status messages, described in the CYBIL System Interface manual.

- For more information, see the NOS/VE Object Code Management manual.

Examples The following example creates a message module containing the status message: +P is not a command.

```
/create_object_library
COL/create_message_module name=a_message_module ..
COL../manual=my_manual
CMM/create_status_message name=cle$unknown..
CMM../_command code=790 severity=error
? +P is not a command.
? **
CMM/end_message_module
COL/
```

CREATE_MODULE
CREOL Subcommand

Purpose Creates a new load module from existing modules and adds it to the module list.

Format

CREATE_MODULE or
CREM
NAME=any
COMPONENT=list of any
GATE=list of any
RETAIN=list of any
STARTING_PROCEDURE=any
PRESET_VALUE=keyword
INCLUDE_BINARY_SECTION_MAPS=boolean
OUTPUT=file
APPLICATION_IDENTIFIER=name
STATUS=status variable

Parameters

NAME or N

Name of the new module.

You use a string value for a module whose name is not an SCL name. Some examples of such module names are: a COBOL module, where a hyphen character (-) may be part of the name, and a C function, where lowercase is significant.

This parameter is required.

COMPONENT or COMPONENTS or C

Component modules of the new module. You can specify a list of files which may be object libraries or object files. Each file name can be followed by a list of modules to be added from that file. If no module names are given, all of the modules are used. You use a string value for a module whose name is not an SCL name.

NOTE

A component module cannot be a command procedure module or a program description module.

The component modules are combined within the new module in the order you list them on the COMPONENT parameter.

This parameter is required. At least one file must be specified.

GATE or *GATES* or *G*

List of additional entry points to be given the gate attribute in the new module.

You use a string value for an entry point whose name is not an SCL name.

If GATE is omitted, the gated entry points in the new module are the entry points gated in the component modules.

RETAIN or *R*

List of additional entry points given the retain attribute.

You use a string value for an entry point whose name is not an SCL name.

If RETAIN is omitted, the new module retains gated entry points, entry points assigned the retain attribute in the component modules, and entry points not referenced by any other component module.

STARTING_PROCEDURE or *SP*

Starting procedure for the new module.

You use a string value for an entry point whose name is not an SCL name.

If STARTING_PROCEDURE is omitted, the starting procedure is the last transfer symbol in the last module specified in the COMPONENT parameter value list.

PRESET_VALUE or *PV*

Specifies text record reduction.

ZERO (Z)

Reduces the number of individual text records in an object module. Reducing the number of records reduces the module loading time.

If PRESET_VALUE is omitted, the number of text records is not reduced.

INCLUDE_BINARY_SECTION_MAPS or *IBSM*

Indicates whether the binary section map is included in the information element for the bound module.

If INCLUDE_BINARY_SECTION_MAPS is omitted, binary section maps are not included.

OUTPUT or *O*

File to which the section map for the new module is written. This file can be positioned.

If OUTPUT is omitted, no section map is written.

APPLICATION_IDENTIFIER or *AI*

Name of the application identifier stored in the module header and included on the application accounting statistics when the software is executed. Only a user with the SYSTEM_ADMINISTRATION or APPLICATION_ADMINISTRATION capability can specify an application identifier.

If an application identifier is placed on a load module, the module is assumed to be a unit-measured application.

If APPLICATION_IDENTIFIER is omitted, no application identifier is assigned to the module.

Remarks

- The new module is not generated until you enter a GENERATE_LIBRARY subcommand. Therefore, the section map for the module is not written on the file specified on the OUTPUT parameter until the module is generated.
- The existing modules to be combined are referred to as the component modules of the new module. The module type of the new module is a bound module because it is created by the combination of other modules.
- If a component module contains an external reference to another component module, CREOL links the modules.
- Although the command adds the new module to the module list, and stores information from the component module headers in the bound module header, it does not add the component modules to the module list. You can display component module information with the subcommand DISPLAY_NEW_LIBRARY or the SCL command DISPLAY_OBJECT_LIBRARY.
- The following entry points are kept in the bound module.
  - The starting procedure entry point for the bound module.
  - Entry points with the gate attribute. (The gate attribute indicates that a procedure executing in a ring within the call bracket of the module can call the entry point.)
  - Entry points with the retain attribute. (The retain attribute indicates the entry point is to be kept in a new module created by combining the module with other modules.)

- Entry points not referenced by any other component module.

- You can assign the gate and retain attributes with the CREOL subcommands CREATE_MODULE or CHANGE_MODULE_ATTRIBUTES. You can also assign the gate attribute within the CYBIL source code (the #GATE attribute on the declaration).

- Do not bind COBOL programs that use CALL and CANCEL into a single module, because CALL will try to overlay a single module that is no longer available.

- For more information, see the NOS/VE Object Code Management manual.

Examples The following subcommand sequence creates a module, displays the module information, then generates a new object library. The new module is named NEW_MODULE and combines modules EXAMPLE and NAND on file OBJ1. When the new library is generated, it writes the section map on file $OUTPUT.

```
COL/create_module name=new_module component=..
COL../((obj1,example,nand)),output=$output
COL/display_new_library module=new_module ..
COL../display_options=(header,component)
 NEW_MODULE          - load module          - 18:05:32 03/28/86
 kind: MI_VIRTUAL_STATE  generator: OBJECT_LIBRARY_GENERATOR
 generator name version: OBJECT LIBRARY GENERATOR V1.1
 components
 ~~~~~~~~~~

   component: EXAMPLE
   created:   18:05:32 1986-03-28
   generator: FORTRAN
   generator name version: FTN
   commentary: VS FORTRAN - level 86063

   component: NAND
   created:   18:05:32 1986-03-28
   generator: FORTRAN
   generator name version: FTN
   commentary: VS FORTRAN - level 86063

COL/generate_library library=my_new_library
 Section map for module  NEW_MODULE            created: 18:05:32  03/28/86
 kind: CODE  length: 6E

   offset: 0          length: 54        module: EXAMPLE     section: EXAMPLE
   offset: 58         length: 16        module: NAND        section: NAND
 kind: BINDING  length: 50

 kind: WORKING STORAGE  length: 180

   offset: 0          length: 0EF       module: EXAMPLE
   offset: 0F0        length: 90        module: NAND
```

```
kind: WORKING STORAGE  length: 88

  offset: 0          length: 78        module: EXAMPLE
  offset: 78         length: 10        module: NAND
```

CREATE_PARAMETER_ASSIST_MESSAGE CREMM Subcommand

Purpose Creates a help message to be displayed when a user enters an incorrect value for the parameter named by this command.

Format **CREATE_PARAMETER_ASSIST_MESSAGE** or **CREPAM**

NAME=name
COLLECT_TEMPLATE_UNTIL=string
STATUS=status variable

Parameters **NAME** or **N**

Name of the command parameter for which the assist message is defined. If the command parameter has aliases, you must specify the first name listed in the PDT or PROC header. This parameter is required.

COLLECT_TEMPLATE_UNTIL or *CTU*

Specifies the termination string used in collection of the template of the parameter assist message. If the COLLECT_TEMPLATE_UNTIL parameter is omitted, the string '**' is assumed.

Remarks For more information, see the NOS/VE Object Code Management manual.

Examples The following creates a parameter assist message.

```
CMM/create_parameter_assist_message name=name
? The NAME parameter must specify the first name
? listed in the command parameter
? descriptor table.
? **
CMM/
```

CREATE_PARAMETER_HELP_MESSAGE CREMM Subcommand

Purpose Creates a help message for a parameter. The help message is displayed when a user requests help for the parameter named by this command.

Format

CREATE_PARAMETER_HELP_MESSAGE or **CREPHM**

NAME=name
COLLECT_TEMPLATE_UNTIL=string
STATUS=status variable

Parameters

NAME or **N**

Name of the command parameter for which the help message is defined. If the command parameter has aliases, you must specify the first name listed in the PDT or PROC header. This parameter is required.

COLLECT_TEMPLATE_UNTIL or *CTU*

Specifies the termination string used in collection of the template of the parameter help message. If the COLLECT_TEMPLATE_UNTIL parameter is omitted, the string '**' is assumed.

Remarks For more information, see the NOS/VE Object Code Management manual.

Examples The following creates a parameter help message.

```
CMM/create_parameter_help_message name=name
? The NAME parameter specifies the command
? parameter for which you are defining
? a help message.
? **
CMM/
```

CREATE_PARAMETER_PROMPT_MESSAGE CREMM Subcommand

Purpose Creates the prompt message that elicits a value for the command parameter named by this command.

Format

CREATE_PARAMETER_PROMPT_MESSAGE or **CREPPM**
NAME=name
COLLECT_TEMPLATE_UNTIL=string
STATUS=status variable

Parameters

NAME or **N**

Name of the command parameter for which the prompt message is defined. If the command parameter has aliases, you must specify the first name listed in the PDT or PROC header. This parameter is required.

COLLECT_TEMPLATE_UNTIL or *CTU*

Specifies the termination string used in collection of the template of the parameter prompt message. If the COLLECT_TEMPLATE_UNTIL parameter is omitted, the string '**' is assumed.

Remarks NOS/VE allows you to name time zones, months, and days in any natural language. The conventions for defining a message module containing these identifiers are as follows:

- To specify an identifier (name) for a time zone, use the following rules:

  The seed name for the message module is TIME_ZONES. Thus, the name of the module specified in the CREMM command for the US_ENGLISH language is TIME_ZONES$US_ENGLISH.

  The parameter prompt message is used to define the identifier for a time zone.Use as the NAME parameter a name with the following format:

  st$hours_digits$minutes_digits or
  dst$hours_digits$minutes_digits

  where st or dst is STANDARD_TIME or DAYLIGHT_SAVING_TIME, hours_digits represents the hours from Coordinated Universal Time (formerly known as Greenwich Mean Time), and minutes_digits represents

the minutes offset of the time zone. If the time zone's hours from Coordinated Universal Time is negative, an underscore (_) must precede the hours_digits value. If the minutes offset of the time zone is negative, an underscore must precede the minutes_digits value. The minutes_digits value is included in the name only if the time zone's minutes offset is not zero.

- Use as the message template text the following format:

  full_identifier, abbreviated_identifier

  If the abbreviated identifier is omitted, it is formed from the first characters of each word of the full identifier.

- To specify an identifier for months and days in natural languages other than US_ENGLISH or ENGLISH, use the following rules. The languages currently supported on NOS/VE are:

DANISH	ITALIAN
DUTCH	NORWEGIAN
ENGLISH	PORTUGESE
FINNISH	SPANISH
FLEMISH	SWEDISH
FRENCH	US_ENGLISH
GERMAN	

  The mechanism for adding such support for other languages is the message module.

  1. The seed name for the message module is MONTHS_AND_DAYS. Thus, the name of the module specified in the CREMM command for the French language is MONTHS_AND_DAYS$FRENCH.

  2. The module must contain specifications for all months of the year and days of the week.

  3. The parameter prompt message defines the name for each month and day. Use as the NAME parameter the name of the month or day in English. As the message template, use text in the following format:

full_name, abbreviated_name

If the abbreviated name is omitted, the first three characters of the full name are used.

4. Once a MONTHS_AND_DAYS module for a particular language has been referenced in a job, modifying the module on the object library or adding a different object library with a MONTHS_AND_DAYS module for the same language to your command list will have no effect on the current job.

- For more information, see the NOS/VE Object Code Management manual.

Examples

- The following creates a parameter prompt message.

```
CMM/create_parameter_prompt_message name=name
? Enter the first name in the PDT or PROC
? header for the command parameter for
? which the prompt message is being defined.
? **
CMM/
```

- The following creates a parameter prompt for time zones in US_ENGLISH.

```
/create_object_library
COL/create_message_module ..
COL../name=time_zones$us_english
CMM/create_parameter_prompt_message ..
CMM../name=standard_time$0
? Coordinated Universal Time
? **
CMM/create_parameter_prompt_message ..
CMM../name=standard_time$_6
? Central Standard Time
? **
CMM/create_parameter_prompt_message ..
CMM../name=daylight_saving_time$_6
? Central Daylight Saving Time, CDT
? **
CMM/end_message_module
COL/generate_library library=my_times
COL/quit
/
```

- The following extracts the released French month and day names.

```
/create_object_library
COL/add_module $system.osf$command_library ..
COL../module=months_and_days$french
COL/generate_library $local.months_days$..
COL../french format=message_module
COL/quit
```

The example DAYS_MONTHS_AND_TIME_ZONES in the online NOS/VE Examples manual demonstrates the definition of message modules for time zone, month names, and day names.

CREATE_PROGRAM_DESCRIPTION CREOL Subcommand

Purpose Defines a program description module and adds it to the module list.

Format

CREATE_PROGRAM_DESCRIPTION or
CREPD
NAME=list of any
FILE=list of any
LIBRARY=list of any or *keyword*
MODULE=list of any
STARTING_PROCEDURE=any
LOAD_MAP=any
LOAD_MAP_OPTION=list of keyword
TERMINATION_ERROR_LEVEL=keyword
PRESET_VALUE=keyword
STACK_SIZE=integer
ABORT_FILE=any
DEBUG_INPUT=any
DEBUG_OUTPUT=any
DEBUG_MODE=boolean
AVAILABILITY=keyword
SCOPE=keyword
LOG_OPTION=keyword
MERGE_OPTION=keyword
APPLICATION_IDENTIFIER=name
ARITHMETIC_OVERFLOW=boolean
ARITHMETIC_LOSS_OF_SIGNIFICANCE=boolean
DIVIDE_FAULT=boolean

EXPONENT_OVERFLOW=boolean
EXPONENT_UNDERFLOW=boolean
FP_INDEFINITE=boolean
FP_LOSS_OF_SIGNIFICANCE=boolean
INVALID_BDP_DATA=boolean
STATUS=status variable

Parameters

NAME or NAMES or N

List of program names. The first name is the module name. Any subsequent names are aliases. A command reference can specify the module by its module name or by an alias. This parameter is required.

FILE or *FILES* or *F*

List of object files or object library files to be unconditionally loaded when the program is executed. A file value is evaluated when the library is generated.

Path values containing $FAMILY, $USER, or $SYSTEM elements can be supplied as strings to be evaluated when the program description is used.

LIBRARY or *LIBRARIES* or *L*

List of library files added to the program library list when the program is executed. A file value is evaluated when the object library is generated.

Path values containing $FAMILY, $USER, or $SYSTEM elements can be supplied as strings to be evaluated when the program description is used.

The keyword OSF$TASK_SERVICES_LIBRARY specifies the system table, and keyword OSF$CURRENT_LIBRARY represents the library containing the program description.

MODULE or *MODULES* or *M*

List of modules loaded from the program library list when the program is executed.

You use a string value for a module whose name is not an SCL name.

If MODULE is omitted, no additional modules are loaded when the program is executed.

STARTING_PROCEDURE or *SP*

Name of the entry point where execution begins.

You use a string value for an entry point whose name is not an SCL name.

If STARTING_PROCEDURE is omitted, the last transfer symbol loaded is used.

LOAD_MAP or *LM*

File on which the load map is written. A file value is evaluated when the library is generated.

Path values containing $FAMILY, $USER, or $SYSTEM elements can be supplied as strings to be evaluated when the program description is used.

This file can be positioned.

LOAD_MAP_OPTION or *LOAD_MAP_OPTIONS* or *LMO*

Set of one or more keywords indicating the information included in the load map. Options are:

NONE

No load map is written.

SEGMENT (S)

Segment map.

BLOCK (B)

Block map.

ENTRY_POINT (EP)

Entry point map.

CROSS_REFERENCE (CR)

Entry point cross-reference.

TERMINATION_ERROR_LEVEL or *TEL*

Error level that terminates program loading. Options are:

WARNING (W)

Warning, error, or fatal error.

ERROR (E)

Error or fatal error only.

FATAL (F)

Fatal error only.

PRESET_VALUE or *PV*

Value stored in all uninitialized words of program space. Options are:

ZERO (Z)

All zeros.

FLOATING_POINT_INDEFINITE (FPI)

Floating-point indefinite value.

INFINITY (I)

Floating-point infinite value.

ALTERNATE_ONES (AO)

Alternating 0 and 1 bits. The leftmost (highest order) bit is 1.

STACK_SIZE or *SS*

Maximum number of bytes in the run-time stack. The program uses the run-time stack for procedure call linkages and local variables.

ABORT_FILE or *AF*

File containing Debug commands to be processed if the program aborts. The commands are used only if the program is not executed in Debug mode. A file value is evaluated when the object library is generated.

Path values containing $FAMILY, $USER, or $SYSTEM elements can be supplied as strings to be evaluated when the program description is used. This file can be positioned.

DEBUG_INPUT or *DI*

File containing Debug commands. The commands are read only if the program is executed under control of Debug (refer to DEBUG_MODE parameter). This file can be positioned. A file value is evaluated when the object library is generated.

Path values containing $FAMILY, $USER, or $SYSTEM elements can be supplied as strings to be evaluated when the program description is used.

DEBUG_OUTPUT or *DO*

File on which Debug output is written. Output is written only if the program is executed in Debug mode. This file can be positioned. A file value is evaluated when the object library is generated.

Path values containing $FAMILY, $USER, or $SYSTEM elements can be supplied as strings to be evaluated when the program description is used.

DEBUG_MODE or *DM*

Indicates whether the program is to be run under the control of Debug. (For information on using Debug, refer to the program's specific source language manual). Options are:

ON

Program executed under control of the Debug program.

OFF

Program executed without the Debug program.

AVAILABILITY or *A*

Specifies whether the program description is made known to users as a command or not. Options are:

ADVERTISED (A)

The program description appears in the output produced by the DISPLAY_COMMAND_LIST_ENTRY command (and in other similar situations).

HIDDEN (H)

The program description is suppressed from the output produced by DISPLAY_COMMAND_LIST_ENTRY command (and in other similar situations).

Omission causes ADVERTISED to be used.

SCOPE or *S*

Reserved for future use.

LOG_OPTION or *LO*

Reserved for future use.

MERGE_OPTION or *MO*

Indicates where the program description module is inserted in the module list. Options are:

ADD (A)

Added to the end of the module list.

REPLACE (R)

Replaces the module with the same name in the module list, if one exists.

COMBINE (C)

Added to the end of the module list if a module of the same name does not exist; replaces the module if it does exist.

If MERGE_OPTION is omitted, COMBINE is used.

APPLICATION_IDENTIFIER or *AI*

Name of the application identifier stored in the module header and included on the application accounting statistics when the software is executed. Only a user with the SYSTEM_ADMINISTRATION or APPLICATION_ADMINISTRATION capability can specify an application identifier.

If APPLICATION_IDENTIFIER is omitted, no application is identified with the program description.

ARITHMETIC_OVERFLOW or *AO*

This parameter specifies whether or not the hardware condition ARITHMETIC_OVERFLOW causes an interrupt. Valid specifications are:

ON

ARITHMETIC_OVERFLOW is enabled. The condition causes an interrupt.

OFF

ARITHMETIC_OVERFLOW is disabled. The condition does not cause an interrupt.

ARITHMETIC_LOSS_OF_SIGNIFICANCE or *ALOS*

This parameter specifies whether or not the hardware condition ARITHMETIC_LOSS_OF_SIGNIFICANCE causes an interrupt. Valid specifications are:

ON

ARITHMETIC_LOSS_OF_SIGNIFICANCE is enabled. The condition causes an interrupt.

OFF

ARITHMETIC_LOSS_OF_SIGNIFICANCE is disabled. The condition does not cause an interrupt.

DIVIDE_FAULT or *DF*

This parameter specifies whether or not the hardware condition DIVIDE_FAULT causes an interrupt. Valid specifications are:

ON

DIVIDE_FAULT is enabled. The condition causes an interrupt.

OFF

DIVIDE_FAULT is disabled. The condition does not cause an interrupt.

EXPONENT_OVERFLOW or *EO*

This parameter specifies whether or not the hardware condition EXPONENT_OVERFLOW causes an interrupt. Valid specifications are:

ON

EXPONENT_OVERFLOW is enabled. The condition causes an interrupt.

OFF

EXPONENT_OVERFLOW is disabled. The condition does not cause an interrupt.

EXPONENT_UNDERFLOW or *EU*

This parameter specifies whether or not the hardware condition EXPONENT_UNDERFLOW causes an interrupt. Valid specifications are:

ON

EXPONENT_UNDERFLOW is enabled. The condition causes an interrupt.

OFF

EXPONENT_UNDERFLOW is disabled. The condition does not cause an interrupt.

FP_INDEFINITE or *FPI* or *FI*

This parameter specifies whether or not the hardware condition FP_INDEFINITE causes an interrupt. Valid specifications are:

ON

FP_INDEFINITE is enabled. The condition causes an interrupt.

OFF

FP_INDEFINITE is disabled. The condition does not cause an interrupt.

FP_LOSS_OF_SIGNIFICANCE or *FPLOS* or *FLOS*

This parameter specifies whether or not the hardware condition FP_LOSS_OF_SIGNIFICANCE causes an interrupt. Valid specifications are:

ON

FP_LOSS_OF_SIGNIFICANCE is enabled. The condition causes an interrupt.

OFF

FP_LOSS_OF_SIGNIFICANCE is disabled. The condition does not cause an interrupt.

INVALID_BDP_DATA or *IBDPD* or *IBD*

This parameter specifies whether or not the hardware condition INVALID_BDP_DATA causes an interrupt. Valid specifications are:

ON

INVALID_BDP_DATA is enabled. The condition causes an interrupt.

OFF

INVALID_BDP_DATA is disabled. The condition does not cause an interrupt.

Remarks

- You can execute the program described by the program description module with a command reference that specifies the module.
- Except where otherwise noted, omitting a parameter from the CREATE_PROGRAM_DESCRIPTION subcommand omits a corresponding attribute from the program description. This causes the corresponding job default program attribute value to be used when the program is executed. You can display the job default attributes by entering the DISPLAY_PROGRAM_ATTRIBUTES command.
- For more information, see the NOS/VE Object Code Management manual.

Examples

- The following subcommand creates a program description for a FORTRAN program.

```
COL/create_program_description ..
COL../name=fortran_program file=$local.lgo ..
COL../library='$local.flf$library'
COL/
```

A name call command that executes module FORTRAN_PROGRAM loads all modules in file $LOCAL.LGO and adds the object library $LOCAL.FLF$LIBRARY to the program library list.

The value $LOCAL.FLF$LIBRARY is specified as a string instead of a file reference because this results in the string $LOCAL.FLF$LIBRARY being evaluated

each time the program description is executed. If $LOCAL.FLF$LIBRARY (a file reference value) is specified, then a value such as

```
:$SYSTEM.$SYSTEM.FORTRAN.FLF$LIBRARY.35
```

is stored in the program description when the object library is generated. Since this references a specific cycle of this FORTRAN library, the program description must be updated if a new version (i.e., a different cycle number) of this FORTRAN library is made available.

- The following sequence demonstrates using OSF$CURRENT_LIBRARY in a program description. The NAME parameter on the program description defines SHOW_OFF and SHOO as two aliases for the FORTRAN program P9939. The LIBRARY parameter specifies that the library on which this program description resides is to be added to the program library list when the program is executed.

  Thus, if the object library is copied to another file, the program description does not have to be updated. The program description always specifies the library on which it resides.

```
/collect_text ftn_pgm
ct?       program p9939
ct?       print *,'In P9939'
ct?       end
ct?**
/fortran input=ftn_pgm
/create_object_library
COL/add_module library=lgo
COL/create_program_description name=..
COL../(show_off shoo) starting_procedure=p9939 ..
COL../library=osf$current_library
COL/generate_library library=my_lib
COL/quit
/my_lib.show_off
In P9939
/copy_file input=my_lib output=diff_lib
/diff_lib.show_off
In P9939
/
```

CREATE_STATUS_MESSAGE CREMM Subcommand

Purpose Creates the description of a status message.

Format

CREATE_STATUS_MESSAGE or
CRESM
NAME=name
CODE=integer
IDENTIFIER=string
SEVERITY=keyword
COLLECT_TEMPLATE_UNTIL=string
STATUS=status variable

Parameters

NAME or **N**

Specifies the condition identifier. This parameter is required.

CODE or **C**

Specifies the status condition code. If the CODE parameter is less than or equal to 16,777,215, the IDENTIFIER parameter must be specified and combined with CODE to form the condition code. If CODE is greater than 16,777,215, it represents the complete status condition code. The IDENTIFIER parameter, if specified, is ignored.

Codes 0 through 9,999 for every possible product identifier are reserved for Control Data use. Codes 10,000 through 19,999 for every possible product identifier are for user-developed products. The remainder of each range is reserved for future use.

The CODE parameter is required.

IDENTIFIER or *I*

Reserved.

SEVERITY or *S*

Specifies the severity level of the status condition. Omission causes ERROR to be assumed. Options are:

NON_STANDARD (N)

Non-standard condition. This flags non-standard extensions to the language specification.

DEPENDENT (D)

Dependent condition. This flags machine dependent usage in code. It is intended primarily for use by the implementation language (CYBIL), but other products with similar needs may also use it.

INFORMATIVE (I)

Information condition. These messages report conditions encountered during command processing that do not cause incorrect or incomplete operation of a command.

WARNING (W)

Warning condition. These messages report conditions encountered during command processing that may have caused incorrect or incomplete operation of a command or of subsequent commands.

ERROR (E)

Error condition. These messages report that the last command was not completed correctly. By default, a batch job is terminated. For an interactive session, additional input is requested from the user to direct continued job processing.

FATAL (F)

Fatal condition. These messages report that the last command or subcommand was not completed correctly. Subsequent processing is usually provided to discover additional problems.

CATASTROPHIC (C)

Catastrophic condition. These messages report that the last command or subcommand was not completed correctly. No further processing for the requested function is possible.

COLLECT_TEMPLATE_UNTIL or *CTU*

Specifies the termination string used in collection of the template of the status message. If the COLLECT_TEMPLATE_UNTIL parameter is omitted, the string '**' is assumed.

Remarks For more information, see the NOS/VE Object Code Management manual.

Examples See the NOS/VE Object Code Management manual for a detailed example.

DELETE_MODULE CREOL Subcommand

Purpose Deletes one or more modules from the module list.

Format

DELETE_MODULE or
DELETE_MODULES or
DELM
MODULE=list of range of any
STATUS=status variable

Parameters

MODULE or **MODULES** or **M**

Modules deleted. If ALL is specified, all modules in the module list are deleted. This parameter is required.

Remarks For more information, see the NOS/VE Object Code Management manual.

Examples The following session generates a new object library from a subset of the modules in an existing object library.

```
/create_object_library
COL/add_modules library=old_library
COL/delete_module (sort4,merge5)
COL/generate_library library=new_library
COL/quit
/
```

The object library generated on file NEW_LIBRARY contains all modules from file OLD_LIBRARY except modules SORT4 and MERGE5.

DISPLAY_NEW_LIBRARY CREOL Subcommand

Purpose Displays information about modules in the module list.

Format

DISPLAY_NEW_LIBRARY or
DISNL
MODULE=list of range of any
DISPLAY_OPTION=list of keyword
OUTPUT=file
ALPHABETICAL_ORDER=boolean
STATUS=status variable

Parameters

MODULE or *MODULES* or *M*

List of modules for which information is displayed.

You use a string value for a module whose name is not an SCL name. Some examples of such module names are: a COBOL module, where a hyphen character (-) may be part of a name, and a C function, where lowercase is significant.

If MODULE is omitted, information for all modules in the module list is displayed.

DISPLAY_OPTION or *DISPLAY_OPTIONS* or *DO*

Set of keywords indicating the information displayed in addition to the module type and name. Options are:

NONE

No information other than the module type and name.

DATE_TIME (DT)

Creation date and time.

ENTRY_POINT (EP)

Entry point names.

HEADER (H)

Module header information. This includes the:

- Module type, name, creation date and time, kind, generator, generator name version, and commentary.

- Formal parameters, availability, scope, and log option for SCL command procedures.
- Entire program description, availability, scope, log option, and application identifier (if one has been specified) for program descriptions.
- Natural language for online manuals and message modules.
- The lowest and highest condition codes for message modules that contain status message information.

LIBRARIES or LIBRARY (L)

Local file names within the object text of the modules that are added to the program library list when the module is loaded (i.e., text-embedded libraries).

REFERENCE (R)

External references.

COMPONENT (C)

Module headers of the component modules if the module is a bound module.

ALL

All of the listed options.

If DISPLAY_OPTION is omitted, the default set by the last SET_DISPLAY_OPTIONS subcommand is used. The initial default is DATE_TIME.

OUTPUT or *O*

Output file. This file can be positioned. If OUTPUT is omitted, file $OUTPUT is used.

ALPHABETICAL_ORDER or *AO*

Indicates the display order for the module information. Options are:

TRUE

Alphabetical order by module name.

FALSE

Order in which modules exist on the library or file.

If ALPHABETICAL_ORDER is omitted, FALSE is used.

Remarks

- The DISPLAY_NEW_LIBRARY subcommand displays the contents of the new library that would be generated if the subcommand GENERATE_LIBRARY were entered.
- To change and display the default display options for subsequent DISPLAY_NEW_LIBRARY subcommands, enter a SET_DISPLAY_OPTION subcommand.
- For more information, see the NOS/VE Object Code Management manual.

Examples The following subcommand lists the module header and entry point information for module EXAMPLE.

```
COL/display_new_library example display_options=(n,ep)
 EXAMPLE                          - object module - 15:40:31 1986-04-09
 kind: MI_VIRTUAL_STATE  generator: CYBIL
 generator name version: C180 CYBIL/II 1.0 LEVEL 85302
 commentary: DA=NONE RC=NONE OPT=LOW
 entry points
 ~~~~~~~~~~~~

   EXAMPLE
 starting procedure: EXAMPLE
```

END_APPLICATION_MENU CREAM Subcommand

Purpose Terminates creation of the application and ends the CREATE_APPLICATION_MENU utility session.

Format

END_APPLICATION_MENU or
QUIT or
ENDAM

Parameters None.

Remarks For more information, see the NOS/VE Object Code Management manual.

END_MESSAGE_MODULE CREMM Subcommand

Purpose Terminates creation of the message module and ends the CREATE_MESSAGE_MODULE utility session.

Format

END_MESSAGE_MODULE or
QUI or
QUIT or
ENDMM
CREATE_MODULE=boolean
STATUS=status variable

Parameters *CREATE_MODULE* or *CM*

Specifies whether the message module should be created. If omitted, YES is assumed and the message module is created.

Remarks For more information, see the NOS/VE Object Code Management manual.

GENERATE_LIBRARY CREOL Subcommand

Purpose Generates a new object library using the information in the module list. This subcommand can also write an object file, SCL procedure text file, or CREATE_MESSAGE_MODULE subcommands.

Format

GENERATE_LIBRARY or
GENL
LIBRARY=file
FORMAT=keyword
STATUS=status variable

Parameters

LIBRARY or **L**

File on which the modules are written. This parameter is required.

FORMAT or *F*

Specifies the format written. Options are:

LIBRARY (L)

Object library. Dictionaries are generated, and each object module in the module list is converted to the load module format. A module dictionary is written on the file.

FILE (F)

Object file. All modules in the module list must be object or load modules. All load modules are converted back to object modules. No dictionaries are generated.

SCL_PROC (SP)

SCL procedure text file. All command procedure modules in the module list are written to the file. This option allows command procedures to be edited on libraries.

MESSAGE_MODULE (MM)

Creates a file containing the CREOL subcommands for building message template modules. When the MESSAGE_MODULE parameter is specified, all message modules in the module list are written to the file. This option allows message module definitions to be edited on libraries.

If FORMAT is omitted, LIBRARY is used.

Remarks

- Refer to the CREATE_OBJECT_LIBRARY command for more specific information on which file attributes are created using GENERATE_LIBRARY.
- The GENERATE_LIBRARY subcommand always discards the contents of the module list after it has used it.

- The GENERATE_LIBRARY subcommand requires ACCESS_MODE=(APPEND SHORTEN) to write the file. If this access cannot be obtained, the file is written to a uniquely named file, and the subcommand reports the file name.

  This should be considered when manipulating an object library that is an entry in the command list. The object library remains open while it is in the command list, and the ACCESS_MODE needed by the GENERATE_LIBRARY subcommand cannot be obtained. The examples that follow include one on updating an object library that is in a command list.

- You can reference the library file written using subsequent subcommands within the CREOL session.

- For more information, see the NOS/VE Object Code Management manual.

Examples

- The following sequence generates an object library that contains the modules from object files OBJ1 and OBJ2.

```
/create_object_library
COL/add_module (obj1,obj2)
COL/generate_library $user.library_1
COL/quit
```

- The following sequence extracts the text in a command procedure stored in the object library on file $USER.MY_PROCED. The SCL command COPY_FILE lists the contents of the text file.

```
/create_object_library
COL/add_module library=$user.my_proced ..
COL../module=proc1
COL/generate_library library=text_file ..
COL../format=scl_proc
COL/copy_file input=text_file
PROC proc1
    attach_file $system.library
    detach_file $system.library2
PROCEND proc1
COL/quit
/
```

- The following sequence demonstrates how to update an object library that is in a command list. It makes the object library $USER.MY_PROCED.1 an entry in the command list, extracts a procedure from the object library, edits the procedure, puts the edited procedure on the new object library $USER.MY_PROCED.2, removes the command list entry for $USER.MY_PROCED.1, and adds the command list entry for $USER.MY_PROCED.2.

```
/create_object_library
COL/add_module library=$user.my_proced.1 ..
COL../module=proc1
COL/generate_library library=proc1_source ..
COL../format=scl_proc
COL/edit_file file=proc1_source
   .
   .   "Use EDIT_FILE to make changes.
   .
COL/add_module library=$user.my_proced.1
COL/replace_module library=proc1_source
COL/generate_library library=$user.my_proced.2
COL/quit
/delete_command_list_entry ..
../entry=$user.my_proced.1
/create_command_list_entry ..
../entry=$user.my_proced.2
```

QUIT
CREOL Subcommand

Purpose Ends a CREATE_OBJECT_LIBRARY utility session.

Format **QUIT** or
QUI

Parameters None.

Remarks For more information, see the NOS/VE Object Code Management manual.

REORDER_MODULE CREOL Subcommand

Purpose Changes the order of one or more modules in the module list.

Format

REORDER_MODULE or
REORDER_MODULES or
REOM
MODULE=list of range of any
PLACEMENT=keyword
DESTINATION=any
STATUS=status variable

Parameters

MODULE or **MODULES** or **M**

List of modules in the order the modules are to appear in the module list.

You use a string value for a module whose name is not an SCL name.

This parameter is required.

PLACEMENT or *P*

Indicates whether the ordered modules are placed before or after the module specified on the DESTINATION parameter. Options are:

BEFORE (B)

Modules placed before the destination module.

AFTER (A)

Modules placed after the destination module.

If PLACEMENT is omitted, AFTER is used.

DESTINATION or *D*

Module before or after which the ordered modules are placed.

If DESTINATION is omitted, the location depends on the PLACEMENT parameter value. If PLACEMENT=BEFORE is specified, the modules are placed at the beginning of the module list; if PLACEMENT=AFTER is specified, the modules are placed at the end of the module list.

Remarks

- To reorder modules, list the modules on the MODULE parameter of the subcommand in the order the modules are to appear in the module list. Then specify the location where CREOL is to insert the modules, as reordered, into the module list using the DESTINATION and PLACEMENT parameters.
- For more information, see the NOS/VE Object Code Management manual.

Examples The following subcommand reorders the modules START, MIDDLE, and END and places them at the end of the module list.

```
COL/reorder_module modules=(start,middle,end)
```

REPLACE_MODULE CREOL Subcommand

Purpose Replaces one or more modules in the module list.

Format

REPLACE_MODULE or
REPLACE_MODULES or
REPM
LIBRARY=list of file
MODULE=list of range of any
STATUS=status variable

Parameters

LIBRARY or **LIBRARIES** or **L**

Object files, SCL procedure files, or object library files containing the replacement modules. This parameter is required.

MODULE or *MODULES* or *M*

Replacement modules.

You use a string value for a module whose name is not an SCL name.

If MODULE is omitted, all modules contained in the files specified on the library parameter are used.

Remarks

- The REPLACE_MODULES subcommand can specify object files, SCL procedure files, or object library files. The files are replaced in the order you specify them on the LIBRARY parameter. If you do not want to use all modules in the files, specify the modules to be used on the MODULE parameter.
- If the name of a specified module matches a module name in the module list, the specified module replaces the existing module. If no module exists with the same name, a warning status is returned and the module is not added to the module list.
- A replacement module is always placed in the module list at the same location as the module it replaces.
- The REPLACE_MODULES subcommand does not add modules to the module list. To add modules, enter an ADD_MODULES subcommand. To add and replace modules, enter a COMBINE_MODULES subcommand.
- For more information, see the NOS/VE Object Code Management manual.

Examples The following subcommand uses all modules on file BINARY to replace modules in the current module list.

```
COL/replace_module library=binary
```

SATISFY_EXTERNAL_REFERENCE CREOL Subcommand

Purpose Adds modules to the module list that satisfy external references.

Format

SATISFY_EXTERNAL_REFERENCE or
SATISFY_EXTERNAL_REFERENCES or
SATER
LIBRARY=list of file
STATUS=status variable

Parameters

LIBRARY or LIBRARIES or L

Object library files that are searched for modules containing referenced entry points. The libraries are searched in the order specified on the parameter. This parameter is required.

Remarks

- You should enter the SATISFY_EXTERNAL_REFERENCES subcommand after you have entered the ADD_MODULE, REPLACE_MODULE, COMBINE_MODULE, and CREATE_MODULE subcommands that specify the initial module list so that a single SATISFY_EXTERNAL_REFERENCES subcommand is effective for the entire new object library.

- If none of the procedures in the new object library request that entry points be loaded dynamically, then the SATISFY_EXTERNAL_REFERENCES subcommand ensures that the object library files specified on the subcommand need not be specified in the program library list when a module from the new object library is loaded. The object libraries need not be specified because all modules required from these libraries are part of the new object library.

  For example, if MODA in the module list references FTNMOD1 and FTNMOD2 from file FTNLIB, a SATISFY_EXTERNAL_REFERENCES subcommand that specifies FTNLIB adds FTNMOD1 and FTNMOD2 to the module list. Later, after the new object library is generated, a subcommand to execute MODA need not specify FTNLIB in the program library list. The loader can load modules FTNMOD1 and FTNMOD2 from the same file as MODA.

- To process a SATISFY_EXTERNAL_REFERENCES subcommand, the CREOL utility generates an external reference list and an entry point list for all modules currently in the module list. It then attempts to match each external reference to an entry point. If the entry point to satisfy an external reference is not in the entry point list, CREOL searches the files specified on the subcommand for a module containing the entry point. The files are searched in the order listed on the subcommand.

  If, after searching each specified file, the CREOL utility does not find the entry point, it continues with the next external reference in the list. No abnormal status is returned if an external reference is not matched.

If the entry point is found, the module is added to the end of the module list. When a module is added to the module list, the entry points and external references within the module are also added to the entry point list and external reference list, respectively. Because the external references of the added modules are added to the external reference list, the SATISFY_EXTERNAL_REFERENCES subcommand also attempts to match the added external references.

The process of matching external references continues until reaching the end of the external reference list, when the entry point and external reference lists are discarded.

- The NOS/VE task service library (OSF$TASK_SERVICES_LIBRARY) should not be used on the SATISFY_EXTERNAL_REFERENCES subcommand. If it is, an error status is returned. There is no way to bind the system entry points into a module such that external references to program interfaces FSP$OPEN_FILE, PMP$EXIT, or similar system routines, can be eliminated from the loading process.

- For more information, see the NOS/VE Object Code Management manual.

Examples

The following sequence compiles a FORTRAN source program and then generates an object library. The object library contains the modules from file MY_LGO, and the modules referenced from the FORTRAN run-time and math libraries on files FLF$LIBRARY and MLF$LIBRARY.

```
/fortran input=source binary=my_lgo
/create_object_library
COL/add_module library=my_lgo
COL/satisfy_external_references ..
COL../libraries=(flf$library,mlf$library)
COL/generate_library library=$user.my_library
COL/quit
/
```

SET_DISPLAY_OPTION CREOL Subcommand

Purpose Changes and displays the default display options for subsequent DISPLAY_NEW_LIBRARY subcommands within the CREATE_OBJECT_LIBRARY session.

Format

SET_DISPLAY_OPTION or
SET_DISPLAY_OPTIONS or
SETDO
DISPLAY_OPTION=list of keyword
STATUS=status variable

Parameters *DISPLAY_OPTION* or *DISPLAY_OPTIONS* or *DO*

List of one or more keywords indicating the new default display options. The keywords indicate the information displayed in addition to the module type and name. Options are:

NONE

No information other than the module type and name.

DATE_TIME (DT)

Creation date and time.

ENTRY_POINT (EP)

Entry point names.

HEADER (H)

Module header information. This includes the following:

- Module type, name, creation date and time, kind, generator, generator name version, and commentary.
- Formal parameters, availability, scope, and log option for SCL command procedures.
- Entire program description, availability, scope, log option, and application identifier (if one has been specified for program descriptions).
- Natural language for online manuals and message modules.

- The lowest and highest condition codes for message modules that contain status message information.

LIBRARIES (L)

Local file names within the object text of the modules that are added to the program library list when the module is loaded (for example, text-embedded libraries).

REFERENCE (R)

External references.

COMPONENT (C)

Module headers of the component modules if the module is a bound module.

ALL

All of the listed options.

If DISPLAY_OPTION is omitted, the default display options are displayed without change.

Remarks

- The initial default display option is DATE_TIME.
- For more information, see the NOS/VE Object Code Management manual.

Examples

The following subcommand changes and displays the default display option.

```
COL/set_display_options display_options=..
COL../(date_time,header,entry_point)
-- display option = (DATE_TIME,HEADER,ENTRY_POINT)
COL/
```

Debug 12

ACTIVATE_SCREEN DEBUG Subcommand

Purpose Moves you to screen mode DEBUG from line mode DEBUG.

Format

ACTIVATE_SCREEN or
ACTS
SOURCE_FILES=list of file
STATUS=status variable

Parameters *SOURCE_FILES* or *SOURCE_FILE* or *SF*

Specifies the file containing the source statements of the program to be debugged in screen mode. If this parameter is omitted, DEBUG requests the file name.

Remarks

- To use this command, the FILE_PROCESSOR attribute of each file must contain the name of the compiler that compiled the file. This is done with the CHANGE_FILE_ATTRIBUTE command before you begin the DEBUG session.
- You can enter ACTIVATE_SCREEN anytime during an interactive DEBUG session.
- When ACTIVATE_SCREEN is entered, any existing breaks are deleted and STEP_MODE is turned off. The DEBUG input and DEBUG output files are changed to specific files used by DEBUG for screen mode.
- Once DEBUG is in screen mode, screen mode functions, certain line mode commands, and SCL commands are available. The DEACTIVATE_SCREEN (DEAS) screen mode functions can be used to return to line mode.
- For more information, see the Debug for NOS/VE Usage manual.

Examples The following example changes the DEBUG session from line mode to screen mode. The source program to be debugged in screen mode is $USER.FORT.

```
DB/activate_screen sf=$user.fort
```

CHANGE_DEFAULT DEBUG Subcommand

Purpose Changes one or more default Debug settings.

Format

CHANGE_DEFAULT or
CHANGE_DEFAULTS or
CHAD
MODULE=module or *keyword*
PROCEDURE=procedure or *keyword*
DEBUG_INPUT=file
DEBUG_OUTPUT=file
STATUS=status variable

Parameters *MODULE* or *M*

New default name for the MODULE parameter on subsequent Debug commands. If you specify $CURRENT, the default module is reset to the module that was executing when Debug gained control. If this parameter is omitted, the current default module remains unchanged.

PROCEDURE or *P*

New default name for the PROCEDURE parameter on subsequent Debug commands. If you specify $CURRENT, the default procedure is reset to the procedure that was executing when Debug gained control. If this parameter is omitted, the current default procedure remains unchanged.

DEBUG_INPUT or *DI*

New default file from which Debug commands are read when Debug next gains control. If this parameter is omitted, the current DEBUG_INPUT file remains unchanged. Unless otherwise specified, the initial DEBUG_INPUT file is $COMMAND.

NOTE

Unless a file position is specified in the file reference, the DEBUG_INPUT and DEBUG_OUTPUT files are positioned at the beginning-of-information the first time it is used. The file is not repositioned the next time it is used. Commands are read from the file sequentially. If an end-of-partition or an end-of-file is reached on the input file, program execution resumes.

DEBUG_OUTPUT or *DO*

New default file on which Debug output is written. The change takes effect immediately. Both break report messages and command output are written to this file. If this parameter is omitted, the current DEBUG_OUTPUT file remains unchanged. Unless otherwise specified, the initial DEBUG_OUTPUT file is $OUTPUT.

Remarks For more information, see the Debug for NOS/VE Usage manual.

Examples Read commands from the file DBIN the next time Debug gains control:

```
DB/change_default debug_input=dbin
```

Write output to file $LIST:

```
DB/change_default debug_output=$list
```

Specify the default module name:

```
DB/chad module=main
```

CHANGE_MEMORY DEBUG Subcommand

Purpose Changes the contents of memory starting at the specified address. You can only change values in memory locations for which you have write permission.

Format

CHANGE_MEMORY or
CHAM
ADDRESS=address
VALUE=any
TYPE=keyword
REPEAT_COUNT=integer or *keyword*
STATUS=status variable

Parameters

ADDRESS or *A*

Address of the first byte of memory to be changed.

The form of an address is rsssoooooooo(16) where r is the ring number, sss is the segment number, and oooooooo is the offset from the beginning of the segment. You can get machine addresses from the cross-reference and load maps for your program. This parameter is required.

VALUE or V

New memory value. An integer value completely replaces the contents of eight bytes. A string value is interpreted as a hexadecimal or ASCII string depending on the TYPE parameter. This parameter is required.

TYPE or *T*

Type of data specified by the VALUE parameter. If this parameter is omitted, a string value is assumed to be a hexadecimal value.

ASCII (A)

ASCII string value.

HEX (H)

Hexadecimal string value.

INTEGER (I)

Integer value.

REPEAT_COUNT or *RC*

Number of times the value is to be repeated in memory. If you specify ALL, it repeats the value until the end of the data segment containing the address. If this parameter is omitted, a value of 1 is used.

Remarks

For more information, see the Debug for NOS/VE Usage manual.

Examples

Replace four bytes of memory beginning at location B02200001112(16) with the hexadecimal string '1010aaab':

```
DB/change_memory address=b02200001112(16) ..
DB../value='1010aaab'
```

Replace six bytes of memory beginning at location B02200000055(16) with the ASCII string 'STRING':

```
DB/change_memory address=b02200000055(16) ..
DB../value='string' type=ascii
```

Replace eight bytes of memory beginning at location B02300000223(16) with the integer value 44:

```
DB/change_memory address=b02300000223(16) value=44
```

CHANGE_PROGRAM_VALUE DEBUG Subcommand

Purpose Changes the value of named program variables. Replacement values are entered in the same format as defined in your program, not as they are represented in memory.

Format

CHANGE_PROGRAM_VALUE or
CHAPV
NAME=variable
VALUE=list of value
MODULE=module
PROCEDURE=procedure
RECURSION_LEVEL=integer
RECURSION_DIRECTION=keyword
STATUS=status variable

Parameters

NAME or *N*

Name of the program variable in the source program whose value is to be changed.

VALUE or **V**

New value for the variable. This parameter is required.

MODULE or *M*

Name of the module that contains the variable. If this parameter is omitted, the default module (the module executing when Debug gained control or the module specified by the CHANGE_DEFAULTS command) is used.

PROCEDURE or *P*

Specifies the name of the procedure that contains the variable. If this parameter is omitted, the default procedure (the procedure executing when Debug gained control or the procedure specified by the CHANGE_DEFAULTS command) is used.

NOTE

The following two parameters, RECURSION_LEVEL and RECURSION_DIRECTION, are applicable only when debugging programs written in languages that support recursion (such as CYBIL and PASCAL). The parameter values are ignored for all other languages.

RECURSION_LEVEL or *RL*

Indicates the particular call of a recursive procedure to be used. If RECURSION_DIRECTION specifies FORWARD, the integer 1 specifies the first call, 2 the second call, and so forth; if RECURSION_DIRECTION is omitted or specifies BACKWARD, the integer 1 specifies the most recent call, 2 its predecessor, and so forth;

Recursion applies only to stack variables; it does not apply to variables stored in either a common block or the $STATIC section.

The default value is 1.

RECURSION_DIRECTION or *RD*

Indicates the order in which calls are counted by the RECURSION_LEVEL parameter. The default value is BACKWARD.

FORWARD

The integer 1 specifies the first call, 2 the second call, and so forth.

BACKWARD

The integer 1 specifies the most recent call, 2 its predecessor, and so forth.

Remarks For more information, see the Debug for NOS/VE Usage manual.

Examples The first example refers to the following definition:

```
COMMON /BLK/ DVAL, RVAL, IVAL, ZVAL
DATA DVAL, RVAL, IVAL, ZVAL /20.0D+0, 3.45E+01, 30,
*(+20.0,20.3)/
```

Display initial value of variable DVAL:

```
DB/display_program_value name=dval
dval = 20.
```

Change the value of variable DVAL to 30.0:

```
DB/change_program_value name=dval value=+30.0d+0
```

Display the new value of DVAL:

```
DB/dispv dval
dval = 30.
```

Change the value of variable INDEX:

```
DB/chapv name=index value=63 module=ff_pp
```

Change the value of logical variable VAR:

```
DB/change_program_value var value=true
```

CHANGE_REGISTER DEBUG Subcommand

Purpose Changes the value of the P, A, or X registers that are associated with the program executing when Debug gained control.

Format

CHANGE_REGISTER or
CHANGE_REGISTERS or
CHAR
KIND=keyword
NUMBER=list of range of integer or *keyword*
VALUE=any
TYPE=keyword
STATUS=status variable

Parameters *KIND* or *K*

Specifies the register to change. By default, it changes the P register.

P

Changes the P register.

A

Changes the A registers.

X

Changes the X registers.

NUMBER or *N*

Indicates the A or X registers to change. ALL specifies all registers of the specified kind (A or X).

If KIND is specified, but NUMBER is omitted, the A0 or X0 register is changed.

If the KIND parameter is omitted, the P register is changed and this parameter is ignored.

VALUE or V

New value of the register. This parameter is required.

A P or A register value can be an integer from 0 through 0FFFFFFFFFFF hexadecimal or a string containing up to 12 hexadecimal digits (blanks are ignored). The upper 4 bits are ignored when changing the P register because the ring number in P cannot be changed.

An X register value can be an integer from -7FFFFFFFFFFFFFFF through 7FFFFFFFFFFFFFFF hexadecimal or a string containing up to 16 hexadecimal digits (blanks are ignored) or an ASCII string of up to eight characters.

The upper bits of the register are set to zero if an integer is negative or to 1 if an integer is positive when the value does not fill the register. A string value is left-justified with remaining bytes unchanged.

TYPE or *T*

Type of data specified by the VALUE parameter. If this parameter is omitted, a string value is assumed to be a hexadecimal value and a numeric value is assumed to be an integer.

ASCII (A)

ASCII string value.

HEX (H)

Hexadecimal string value.

INTEGER (I)

Integer value.

Remarks For more information, see the Debug for NOS/VE Usage manual.

Examples Change the current value of the P register to 0A02200004500(16). The upper 4 bits for the ring number are ignored.

```
DB/change_register kind=p value=0a02200004500(16)
```

Change the current value of the X7 register to 'ABCDEFGH':

```
DB/char kind=x number=7 value='abcdefgh' type=ascii
```

$CURRENT_LINE DEBUG Function

Purpose Returns an integer identifying the current line number in the program where Debug has control.

Format **$CURRENT_LINE** or **$CL**

Parameters None.

Remarks For more information, see the Debug for NOS/VE Usage manual.

$CURRENT_MODULE DEBUG Function

Purpose Returns a string identifying the name of the module where execution stopped.

Format **$CURRENT_MODULE** or **$CM**

Parameters None.

Remarks For more information, see the Debug for NOS/VE Usage manual.

$CURRENT_PROCEDURE DEBUG Function

Purpose Returns a string identifying the name of the procedure where execution is stopped.

Format **$CURRENT_PROCEDURE** or **$CP**

Parameters None.

Remarks For more information, see the Debug for NOS/VE Usage manual.

$CURRENT_PVA DEBUG Function

Purpose Returns an integer identifying the process virtual address (PVA) where execution is stopped.

Format
$CURRENT_PVA or
$CPVA

Parameters None.

Remarks For more information, see the Debug for NOS/VE Usage manual.

DELETE_BREAK DEBUG Subcommand

Purpose Deletes one or more break definitions.

Format
DELETE_BREAK or
DELETE_BREAKS or
DELB
BREAK=list of name
STATUS=status variable

Parameters **BREAK** or **BREAKS** or **B**

Specifies the break definitions to be deleted. If the keyword ALL appears in the list of names, all breaks are deleted. This parameter is required.

Remarks For more information, see the Debug for NOS/VE Usage manual.

Examples Delete break definitions B1, B2, and B3:

```
DB/delete_breaks breaks=(b1,b2,b3)
```

Delete all break definitions:

```
DB/delete_breaks all
```

Delete break definition B4:

```
DB/delete_break b4
```

DISPLAY_BREAK DEBUG Subcommand

Purpose Displays specified break definitions. The break name, events, address, and any commands associated with the break are displayed.

Format

DISPLAY_BREAK or
DISPLAY_BREAKS or
DISB
BREAKS=list of name
OUTPUT=file
STATUS=status variable

Parameters *BREAKS* or *BREAK* or *B*

Break definitions to be displayed. If the keyword ALL appears in the list of names, all breaks are displayed. If this parameter is omitted, all breaks are displayed.

OUTPUT or *O*

File on which the break definitions are written. The default file is the current default Debug output file.

Remarks For more information, see the Debug for NOS/VE Usage manual.

Examples Display break definitions B1, B3:

```
DB/display_breaks breaks=(b1,b3)

   -- Break B1
   -- event(s) = execution
   -- location: M=TEST L=16

   -- Break B3
   -- event(s) = eu
   -- range: M=TEST L=18 to M=TEST L=19 BO=15
```

Display all break definitions:

```
DB/display_breaks

   -- Break B1
   -- event(s) = execution
   -- location: M=TEST L=16

   -- Break B2
   -- event(s) = execution
   -- range: M=TEST L=22 to M=test BO=39

   -- Break B3
   -- event(s) = eu
   -- range: M=TEST L=18 to M=TEST L=19 BO=15

   -- Break B4
   -- event(s) = execution
   -- location: M=TEST L=25
```

DISPLAY_CALL DEBUG Subcommand

Purpose Displays information about the dynamic call chain.

Format

DISPLAY_CALL or
DISPLAY_CALLS or
DISC
COUNT=integer or *keyword*
START=integer
DISPLAY_OPTION=list of keyword
OUTPUT=file
STATUS=status variable

Parameters

COUNT or *C*

Number of calls to be displayed. If the keyword ALL is specified or this parameter is omitted, all calls are displayed.

START or *S*

Call in the chain to be displayed first. The integer 1 specifies the most recent call, 2 the predecessor to the most recent call, and so forth. The default value is 1, the most recent call.

DISPLAY_OPTION or *DISPLAY_OPTIONS* or *DO*

Type of information to be displayed. If this parameter is omitted, only calls which are in user code are displayed.

USER_CALLS (UC)

Displays only calls which are in user code.

SYSTEM_CALLS (SC)

Displays only calls which are not part of the user code.

ALL_CALLS (AC)

Displays both user calls and system calls.

VARIABLE_VALUES (VV)

Displays all variables known to the procedure.

OUTPUT or *O*

File on which the call information is written. If this parameter is omitted, the information is written to the current DEBUG_OUTPUT file.

Remarks For more information, see the Debug for NOS/VE Usage manual.

Examples Display all the calls on the call chain beginning with the second most recent call:

```
DB/display_calls start=2

 -- Called from procedure IO1 module IO1 at line 16
    byte offset 12
 -- Called from procedure TEST module TEST at line 54
    byte offset 12
```

Display system code calls on the call chain beginning with the third most recent call:

```
DB/display_calls start=3 display_option=system_calls

 -- Traceback from module MLM$OUTPUT_FLOATING_NUMBER
    byte offset 14C(16)
 -- Called from procedure FLP$SEQ_ACC_LST_OUT module
    FLM$LIST_DIRECTED_IO byte offset 61E(16)
```

Display all user code calls on the call chain, as well as all variables known to the procedure:

```
DB/display_calls display_options=(user_call, ..
DB../variable_values)

 -- Traceback from procedure TEST module TEST at
    line 42
 -- DISPLAY OF ALL VARIABLES IN TEST

        ARAY =  0( 20 OCCURRENCES)
        ARG = 0
        BASE = 2.
        DVAL = 20.
        I = 0
        IVAL = 30
        LOG1 = FALSE
        LOG2 = FALSE
        LOG3 = FALSE
        PIND = 70000000000000000(16)
        RESLT = 0.
        RVAL = 34.5
        X1 = 0
        XPWR = 4094.
        Y1 = 0.
        ZERO = 0.
        ZVAL = (20.,20.3)
```

DISPLAY_DEBUGGING_ENVIRONMENT DEBUG Subcommand

Purpose Displays the following:

- Current defaults for module, procedure, DEBUG_INPUT, DEBUG_OUTPUT
- Total number of breaks you have set and Debug has set
- STEP_MODE values
- Location in your program where execution has stopped

Format

DISPLAY_DEBUGGING_ENVIRONMENT or **DISDE**

DISPLAY_OPTION=list of keyword
OUTPUT=file
STATUS=status variable

Parameters

DISPLAY_OPTION or *DISPLAY_OPTIONS* or *DO*

Type of information to be displayed. If this parameter is omitted, defaults, breaks, STEP_MODE attributes, and user addresses are displayed.

DEFAULTS (D)

Current default values for module, procedure, DEBUG_INPUT, and DEBUG_OUTPUT.

Unless the CHANGE_DEFAULT subcommand has been specified, the default module and procedure is where execution has stopped in your task. The text $CURRENT is output if module or procedure has not been initialized.

BREAKS (B)

Number of breaks you have set, number of breaks currently in use by Debug, and the maximum number of allowed breaks.

STEP_MODE (SM)

Current STEP_MODE attributes.

USER_ADDRESS (UA)

Location where execution has stopped in your program.

ALL

Displays defaults, breaks, STEP_MODE attributes, and the user address.

OUTPUT or *O*

File where the debugging environment display is written. If this parameter is omitted, the current default Debug output file is written.

Remarks

For more information, see the Debug for NOS/VE Usage manual.

Examples Display the number of breaks set, the number of breaks in use by Debug, the maximum number of allowed breaks, and the location where execution has stopped:

```
DB/display_debugging_environment do=(b,ua)

 -- The number of breaks set by the user is 1.
 -- The number of breaks in use by DEBUG is 0.
 -- The number of available breaks is 63.
 -- Execution is currently stopped at B 02E 0000013C
    which, in higher symbolic terms is M=TEST L=36
    BO=12
```

Write defaults, breaks, STEP_MODE attributes, and location where execution has stopped to file FILE1 and returns the status to variable SS:

```
DB/disde do=all output=file1 status=ss
```

Write defaults, breaks, STEP_MODE attributes, and location where execution has stopped to the current default Debug output file:

```
DB/disde

 --  Default module is $CURRENT.
 --  Default procedure is $CURRENT.
 --  Default debug_input file is DBIN
 --  Default debug_output file is $OUTPUT
 --  The number of breaks set by the user is 5
 --  The number of breaks in use by DEBUG is 0.
 --  The number of available breaks is 59.
 --  Step_mode is OFF.
 --  Execution is currently stopped at B 02E 0000013C
     which, in higher symbolic terms is M=TEST L=36
     BO=12
```

DISPLAY_MEMORY DEBUG Subcommand

Purpose Displays information located at a location to which you have read access. The location can be specified by section and module or by address.

Format

DISPLAY_MEMORY or
DISM
ADDRESS=address
SECTION=name
MODULE=module
BYTE_OFFSET=integer
BYTE_COUNT=integer
REPEAT_COUNT=integer or *keyword*
OUTPUT=file
STATUS=status variable

Parameters

ADDRESS or *A*

Address of the first byte of memory to be displayed. If the ADDRESS parameter is omitted, the location must be specified by the SECTION and MODULE parameters.

The address has the format rssso0000000(16) where r is the ring number, sss is the segment number, and oooooooo is the offset from the beginning of the segment. You can use the BYTE_OFFSET parameter to modify the starting address of memory to be displayed. This parameter is required.

SECTION or *SEC*

Memory section containing the data to be displayed.

$BLANK

Section that contains unnamed common.

$BINDING

Section that contains the links to external procedures and the data for the module.

$LITERAL

Section containing the literal data (for example, long constants) of the module.

$STATIC

Section containing the static (not on the run-time stack) variables not explicitly allocated to a named section of the module.

When you use SECTION to specify a location, you must qualify it with the MODULE parameter. You can use the BYTE_OFFSET parameter to modify the starting address of memory to be displayed.

MODULE or *M*

Module containing the data to be displayed. The MODULE parameter cannot be specified unless the SECTION parameter is also specified. If MODULE and SECTION are omitted, the location must be specified by the ADDRESS parameter.

BYTE_OFFSET or *BO*

Offset to the location specified by the SECTION and MODULE parameters or the ADDRESS parameter. If BYTE_OFFSET is omitted, a zero offset is used.

The address generated by adding BYTE_OFFSET to the base address must be within the memory block implied by the base address. The block size is the length of the section when the SECTION parameter is specified, and the length of the segment containing the machine address when the ADDRESS parameter is specified.

BYTE_COUNT or *BC*

Number of bytes in the item to be displayed. The default value is eight bytes.

REPEAT_COUNT or *RC*

Number of items of length BYTE_COUNT to be displayed. If REPEAT_COUNT is omitted, only one item is displayed.

The maximum amount of memory that can be displayed is limited to the block size implied by address (section length for SECTION and segment length for ADDRESS). A large integer causes all memory from the specified address to the end of the memory block to be displayed.

The keyword ALL displays all memory from the specified address to the end of the memory block.

OUTPUT or *O*

File on which the displayed information is written. If OUTPUT is omitted, the display is written to the current Debug output file.

Remarks

- This command allows you to debug your program even when compiler-generated symbol tables are not available, and to display memory areas that do not correspond to program identifiers. Each display line shows the memory contents in hexadecimal and ASCII formats; the relative byte offset from the initial address is also shown.
- The compiler-generated attributes list shows the section name and offset for all variables. To reference static variables, specify the section name and byte offset. To reference variables on the stack, specify the machine address of the stack frame and byte offset.
- To get the address of the stack frame of the procedure executing when Debug got control, display register A1 (see the DISPLAY_REGISTER command description). To get the address of other stack frames, display the save area of the wanted stack frame using the DISPLAY_STACK_FRAME command and get the value of register A1 from that display.
- You can use the DISPLAY_PROGRAM_VALUE command to display program variables when symbol tables are available.
- For more information, see the Debug for NOS/VE Usage manual.

Examples

Display the first three bytes of the literal memory section for module MOD1:

```
DB/display_memory section=$literal module=mod1 ..
DB../byte_count=3
```

Display the first 32 bytes of the memory section DATA1 for module MOD2 as separate items:

```
DB/display_memory sec=data1 module=mod2 rc=4
```

Display the first 200 bytes of memory starting from the specified address:

```
DB/dism a=0b02400000224(16) bo=8 rc=25
```

DISPLAY_PROGRAM_VALUE DEBUG Subcommand

Purpose Displays the value of a program variable.

Format

DISPLAY_PROGRAM_VALUE or
DISPV
NAME=list of variable
MODULE=module
PROCEDURE=procedure
RECURSION_LEVEL=integer
RECURSION_DIRECTION=keyword
TYPE=keyword
VARIANT_SELECTION=list of any
NAME_OPTION=list of keyword
SCOPE=keyword
SECTION=section or *keyword*
OUTPUT=file
STATUS=status variable

Parameters *NAME* or *N*

Name of the program variable in the source program whose value is to be displayed, or the keyword $ALL to display all variables in the procedure. This parameter is required.

The program variable can be one of the following:

- Simple variable or constant name.
- Substring reference.
- Subscripted name.
- Field reference.
- Pointer reference or dereference.

Subscripts can be constants or variables but not expressions. NAME cannot be a substring.

SCL string variables can be used to name long program names. To do this, assign a string containing the identifier to the SCL variable. Then use the SCL variable preceded by a question mark as the value of the NAME parameter.

MODULE or *M*

Name of the module that contains the variable. The default module is the module executing when Debug gained control or the module specified by the CHANGE_DEFAULT subcommand.

PROCEDURE or *P*

Name of the procedure that contains the variable. The default procedure is the procedure executing when Debug gained control or the procedure specified by the CHANGE_DEFAULTS command.

NOTE

The following two parameters, RECURSION_LEVEL and RECURSION_DIRECTION, are applicable only when debugging programs written in languages that support recursion (such as CYBIL and PASCAL). The parameter values are ignored for all other languages.

RECURSION_LEVEL or *RL*

Indicates the particular call of a recursive procedure to be used. If RECURSION_DIRECTION specifies FORWARD, the integer 1 specifies the first call, 2 the second call, and so forth; if RECURSION_DIRECTION is omitted or specifies BACKWARD, the integer 1 specifies the most recent call, 2 its predecessor, and so forth;

Recursion applies only to stack variables; it does not apply to variables stored in either a common block or the $STATIC section.

The default value is 1.

RECURSION_DIRECTION or *RD*

Indicates the order in which calls are counted by the RECURSION_LEVEL parameter. The default value is BACKWARD.

FORWARD

The integer 1 specifies the first call, 2 the second call, and so forth.

BACKWARD

The integer 1 specifies the most recent call, 2 its predecessor, and so forth.

TYPE or *T*

Data representation used for the display.

HEX (H)

Hexadecimal dump. The display includes the variable name, its starting address, and the data displayed as hexadecimal digits and as ASCII characters.

INTEGER (I)

Decimal integer. The data length must be from 1 through 8 bytes. Each element of an array is displayed as a separate integer.

REAL (R)

Floating-point. The data length must be 8 bytes. Each element of an array is displayed as a separate floating-point number.

If TYPE is omitted, the data representation used corresponds to the data type as defined in the program.

VARIANT_SELECTION or *VS*

Selector value specifying the tagless variant to be displayed. The specified value can be an integer, boolean, name or one-character string, but it cannot be a string longer than one character. The value specifies the ordinal of the variant to be displayed.

Debug prompts the user when the VARIANT_SELECTION parameter is required, but has not been supplied.

NAME_OPTION or *NAME_OPTIONS* or *NO*

Qualifies the identifier(s) given for the NAME parameter. Options are:

Omitted ,

There is no default for the NAME_OPTION parameter when a single identifier is specified for the parameter. If $ALL is specified for the NAME parameter, the default for the NAME_OPTION parameter is VARIABLES.

CONSTANTS (C)

The identifier in the source program must be a constant.

VARIABLES (V)

The identifier in the source program must be a variable.

PARAMETERS (P)

The identifier in the source program must be a variable that was passed as a parameter to the default procedure or the procedure specified by the PROCEDURE parameter.

ALL

The identifier in the source program can be either a constant or a variable.

NOTE

NAME_OPTIONS=PARAMETERS cannot be used with the SECTION parameter.

SCOPE or *SCO*

Determines the type of search for identifiers specified by the NAME parameter. Options are:

GLOBAL (G)

The value of the NAME parameter must reference identifier(s) known outside the defining module. The Entry Point Table is searched to locate the identifier(s).

NOTE

GLOBAL cannot be used with the MODULE, PROCEDURE, RECURSION LEVEL, and RECURSION_DIRECTION parameters or with the NAME_OPTION =PARAMETERS.

MODULE (M)

The value of the NAME parameter must reference identifiers(s) defined at the outermost level of the module.

LOCAL (L)

The identifier(s) referenced by the NAME parameter must be defined in the procedure specified by the PROCEDURE parameter or by default.

SECTION or *SEC*

Displays a group of identifiers by specifying the section where they are stored. This parameter is valid only when the value of the NAME parameter is $ALL.

NOTE

The SECTION parameter cannot be used with the RECURSION_LEVEL and RECURSION_DIRECTION parameters or with NAME_OPTION=PARAMETERS or SCOPE=GLOBAL.

OUTPUT or *O*

File where the display information is written. The default is the current Debug output file.

Remarks For more information, see the Debug for NOS/VE Usage manual.

Examples The examples refer to the following definitions:

```
COMMON /BLK/ DVAL, RVAL, IVAL, ZVAL
DATA DVAL, RVAL, IVAL, ZVAL/20.0D+0, 3.45E+01, 30, (+20.0,20.3)/
```

Display the value of DVAL:

```
DB/display_program_value name=dval
dval = 20.
DB/
```

Display the value of RVAL:

```
DB/dispv name=rval
rval = 34.5
DB/
```

Display the value of IVAL:

```
DB/display_program_value ival
ival = 30
DB/
```

DISPLAY_REGISTER DEBUG Subcommand

Purpose Displays the contents of the P, A, or X registers that are associated with the procedure executing when Debug gained control.

Format

DISPLAY_REGISTER or
DISPLAY_REGISTERS or
DISR

KIND=list of keyword
NUMBER=list of range of integer or *keyword*
TYPE=keyword
OUTPUT=file
STATUS=status variable

Parameters

KIND or *K*

Register to be displayed. By default, it displays all registers.

P

Displays the P register.

A

Displays the A registers.

X

Displays the X registers.

NUMBER or *N*

Indicates the A or X registers to display. ALL specifies all registers of the specified kind (A or X).

If the KIND parameter is omitted, the P register is assumed and this parameter is ignored. If KIND is specified, but NUMBER is omitted, the A0 or X0 register is displayed.

TYPE or *T*

Type of data to be displayed. If this parameter is omitted, a string value is assumed to be a hexadecimal value and a numeric value is assumed to be an integer.

ASCII (A)

ASCII string value.

HEX (H)

Hexadecimal string value.

INTEGER (I)

Integer value.

OUTPUT or *O*

File where the display information is written. The default file is the current Debug output file.

Remarks For more information, see the Debug for NOS/VE Usage manual.

Examples Display the contents of the P register in hexadecimal:

```
DB/display_register p

P=B 031 00000040
```

Display the contents of the A8 register in hexadecimal:

```
DB/display_register kind=a number=8 type=hex

A8=B 04E 000004A8
```

Display the contents of the X4, X5, X6, X7, X8, X9, and X10 registers in hexadecimal:

```
DB/disr kind=x number=4..10

X4=70000000   0000000
X5=40019482   53FC0CD1
X6=0000B01B   0000253A
X7=00000000   00000001
X8=00000000   00000064
X9=00000000   00000273
XA=00000000   000000CA
```

DISPLAY_STACK_FRAME DEBUG Subcommand

Purpose Displays the contents of one or more stack frames. Values are displayed in hexadecimal.

Format

DISPLAY_STACK_FRAME or
DISPLAY_STACK_FRAMES or
DISSF
COUNT=integer or *keyword*
START=integer
DISPLAY_OPTION=list of keyword
OUTPUT=file
STATUS=status variable

Parameters

COUNT or *C*

Number of stack frames to be displayed. The keyword ALL displays all stack frames. The default is one stack frame.

START or *S*

Frame on the stack to be displayed first. The integer 1 represents the most recent stack frame, 2 the predecessor of the most recent stack frame, and so forth. By default, the display begins with the most recent stack frame.

DISPLAY_OPTION or *DISPLAY_OPTIONS* or *DO*

Area of the stack frames to be displayed. By default, it displays both the automatic and save areas.

AUTO (A)

Displays the area that contains the automatic (dynamically allocated) variables of the procedure.

SAVE (S)

Displays the area that contains a copy of the registers of the procedure as they existed at the time of a call or trap.

ALL

Displays both the automatic and save areas.

OUTPUT or *O*

File on which the stack frame information is written. The default file is the current DEBUG_OUTPUT file.

Remarks For more information, see the Debug for NOS/VE Usage manual.

Examples Display the save area of the most recent stack frame:

```
DB/display_stack_frame display_option=save

SAVE AREA

P=B 035 00000026          VMID=0
UM=FFF7     UCR=0040     MCR=0000


A0=B 032 00000460       A1=B 032 00000408
A2=B 032 000003C0       A3=B 030 00000000
A4=B 032 00000390       A5=B 02F 00000020
A6=B 02E 00000000       A7=B 02F 00000000
A8=B 00F 00000018       A9=B 032 00000630
AA=B 032 00000A30       AB=F FFF 80000000
AC=F FFF 80000000       AD=B 032 00001058
AE=F FFF 80000000       AF=B 00B 000557F8

X0=0000B01D  00020060       X1=00000000  00000000
X2=0000FFFF  80000000       X3=000007FF  FFFFFFFF

X4=00000000  10000000       X5=00000000  00000008
X6=00000000  0000000D       X7=00000000  0000001D
X8=00000000  00000000       X9=00000000  00000008
XA=00000000  00000300       XB=00000000  00000000
XC=00000000  00000001       XD=00000000  00000022
XE=00000000  00010040       XF=00000000  0000004E
```

Display the automatic and save areas of the most recent stack frame:

```
DB/dissf  count=1

STACK FRAME 001         SEGMENT=032
00000000     00000000  00000000
00000008     00000000  00000000
00000010     30300000  00C0FFFF     00
00000018     80000000  00000000
00000020     B032B031  00000000      2 1
00000028     0000B01D  0009B346              F
00000030     0000B032  00000430        2     0
00000038     0040B032  00000400      @ 2
00000040     FF77B032  000003C0      w 2
00000048     FFFCB01B  00020F78              x
00000050     0000B032  00000390        2

SAVE AREA

P=B 035 00000026          VMID=0
UM=FFF7     UCR=0040     MCR=0000


A0=B 032 00000460       A1=B 032 00000408
A2=B 032 000003C0       A3=B 030 00000000
A4=B 032 00000390       A5=B 02F 00000020
A6=B 02E 00000000       A7=B 02F 00000000
A8=B 00F 00000018       A9=B 032 00000630
AA=B 032 00000A30       AB=F FFF 80000000
AC=F FFF 80000000       AD=B 032 00001058
AE=F FFF 80000000       AF=B 00B 000557F8

X0=0000B01D  00020060       X1=00000000  00000000
X2=0000FFFF  80000000       X3=000007FF  FFFFFFFF

X4=00000000  10000000       X5=00000000  00000008
X6=00000000  0000000D       X7=00000000  0000001D
X8=00000000  00000000       X9=00000000  00000008
XA=00000000  00000300       XB=00000000  00000000
XC=00000000  00000001       XD=00000000  00000022
XE=00000000  00010040       XF=00000000  0000004E
```

$MEMORY
DEBUG Function

Purpose Returns the contents of memory which can be used as input to the DISPLAY_MEMORY or CHANGE_MEMORY commands. You can display memory which is the object of a pointer in memory if the pointer is contained in a register.

Format

$MEMORY or
$MEM
(address
integer
keyword)

Parameters

address

Specifies the process virtual address. This parameter is required.

integer

The number of bytes to return. If kind is an integer, number must be in the range of 1 through 8. If kind is a string, number must be in the range of 1 through 256. If number is omitted, 6 bytes are returned.

keyword

The type of value to return. If kind is integer, a hexadecimal integer with radix is returned. If kind is a string, a string is returned. If kind is omitted, an integer is returned.

Remarks For more information, see the Debug for NOS/VE Usage manual.

$PROGRAM_VALUE
DEBUG Function

Purpose Returns the value of the program element which is specified as the name parameter. Additional parameters for module, procedure, recursion level, and recursion direction can be specified to fully identify the named variable.

The $PROGRAM_VALUE function allows you to incorporate the values of program variables in SCL statements in order to enhance debugging capabilities.

Format

$PROGRAM_VALUE or
$PV
(program_value
module
procedure
integer
keyword)

Parameters

program_value

Name of the program element whose value is to be displayed. This parameter is required. Values can be one of the following types:

- Simple variable
- Subscripted name
- Field reference
- Pointer reference

The named variable must be used in your program.

Because names can be long, SCL string variables can be used as aliases for them. To do this, assign the SCL variable to a string containing the identifier. Then use the SCL variable preceded by a question mark as the value of the name parameter.

module

Name of the module that contains the element specified by the name parameter. Omission causes the module executing when Debug gained control or the module specified by the CHANGE_DEFAULT subcommand to be used.

procedure

Name of the procedure that contains the element specified by the name parameter. If you specify a procedure that is not in the active call chain, its automatic variables cannot be used because it has no stack frame. Omission causes the procedure executing when Debug gained control to be used if a module name is not specified. Otherwise, there

is no default procedure when a module name is specified and a procedure name is not specified; the element specified by the name parameter must exist at the module level.

integer

The particular call of a recursive procedure to be used. It must be a positive integer greater than zero. If the recursion direction parameter specified the keyword FORWARD, a value of 1 is the first call, 2 is the second call (the one called by the first call), and so on. If the recursion direction parameter is BACKWARD, 1 is the most recent call, 2 is the predecessor, and so on.

Omission causes a value of 1 to be used.

keyword

Order in which calls to a recursive procedure are searched. It controls how the value of the recursion_level parameter is interpreted. It can be one of the following keywords:

FORWARD or F

If the RECURSION_LEVEL parameter specifies that the first call to the procedure is used, a 2 specifies the second call, and so on.

BACKWARD or B

If the RECURSION_LEVEL parameter specifies that the most recent call to the procedure is used, a 2 specifies its predecessor, and so on.

Omission causes BACKWARD to be used.

Remarks For more information, see the Debug for NOS/VE Usage manual.

Examples

```
DB/set_break name=b1 line=23 command= ..
DB../'if $program_value(index) <45 then; run; ifend'
```

QUIT
DEBUG Subcommand

Purpose Ends the Debug session and returns control to NOS/VE. The session is terminated immediately; the program is not executed to completion.

Format

QUIT or
QUI
STATUS=status variable

Remarks For more information, see the Debug for NOS/VE Usage manual.

$REGISTER
DEBUG Function

Purpose Returns the contents of a specified register in hexadecimal integer format, including radix. $REGISTER is useful when specified for the ADDRESS parameter value on the DISPLAY_MEMORY and CHANGE_MEMORY commands.

Format

$REGISTER or
$REG
(keyword
integer)

Parameters

keyword

The type of register the value is returned from. P specifies a P register, A specifies an A register, and X specifies an X register.

integer

Specifies the register number the value is returned from.

Remarks For more information, see the Debug for NOS/VE Usage manual.

RUN
DEBUG Subcommand

Purpose Begins or resumes program execution once Debug has gained control. Execution continues until Debug again gains control. If the program has run to completion, entering the RUN command terminates program execution.

Format

RUN
STATUS=status variable

Remarks

- Execution begins at the instruction whose address is stored in the P register of the program when the event that caused Debug to gain control occurred.
- If the P register points to the instruction that caused the event (such as division by zero), the same event will occur immediately after entering the RUN command. In this case, you must change the value in the P register with the CHANGE_REGISTER command or change the value of one of the operands with the CHANGE_PROGRAM_VALUE command before entering the RUN command.
- When Debug processes the RUN command, all previously created SCL blocks (except SET_BREAK command information and the name of the current DEBUG_INPUT and DEBUG_OUTPUT files) are lost. This means that all information about SCL commands, such as if-then blocks or while-for loops that span RUN commands are lost. Global variables must be recreated with XREF.
- For more information, see the Debug for NOS/VE Usage manual.

Examples The following sequence recreates the variable COUNT:

```
DB/crev  count  kind=integer  scope=job  v=0
DB/setb  b=one  l=one
DB/run  "BREAK ONE"
```

By specifying SCOPE=JOB, the variable COUNT will be retained past the RUN command.

SET_BREAK
DEBUG Subcommand

Purpose Defines a break.

Format

SET_BREAK or
SETB
BREAK=name
EVENT=list of name
LINE=integer
STATEMENT=integer
STATEMENT_LABEL=statement_label
NAME=variable

SECTION=section or *keyword*
MODULE=module
PROCEDURE=procedure
ENTRY_POINT=entry_point
ADDRESS=address
BYTE_OFFSET=integer
BYTE_COUNT=integer
COMMANDS=string
STATUS=status variable

Parameters

BREAK or *B*

Name of the break. By default, Debug assigns a unique name and displays the name assignment to the user.

The name is used to reference the break definition in the DISPLAY_BREAK and DELETE_BREAK commands. The name is displayed in the break report message when the break occurs.

A break cannot be named ALL. The break name must not contain the character '$'. The form is:

EVENT or *EVENTS* or *E*

One or more events that will cause the break. If you specify more than one event, the break occurs for any of the events.

ARITHMETIC_OVERFLOW (AO)

Breaks when an arithmetic overflow occurs on an instruction in the specified address range. The P register points to the instruction that caused the overflow.

ARITHMETIC_SIGNIFICANCE (AS)

Breaks when arithmetic significance is lost on an instruction in the specified address range. The P register points to the instruction that caused the loss of significance.

BRANCH (B)

Breaks before either a branch to or a return from any location in the specified address range occurs.

CALL (C)

Breaks before a subprogram call occurs to any address in the specified address range.

DIVIDE_FAULT (DF)

Breaks when division by zero occurs in an instruction in the specified address range. The P register points to the instruction that caused the division by zero.

EXECUTION (E)

Breaks before the instruction in the specified address range is executed.

If the address is specified by the line number, not every line is usable. For example, breaks cannot be set at ENDIF statements because it is not obvious when control reaches them.

EXPONENT_OVERFLOW (EO)

Breaks when an exponent overflow occurs in an instruction in the specified address range. The P register points to the instruction following the one that caused the overflow.

EXPONENT_UNDERFLOW (EU)

Breaks when an exponent underflow occurs in an instruction in the specified address range. The P register points to the instruction following the one that caused the underflow.

FLOATING_POINT_INDEFINITE (FPI)

Breaks when the result of a floating-point operation is indefinite in an instruction in the specified address range. The P register points to the instruction following the one that caused the results to be indefinite.

FLOATING_POINT_SIGNIFICANCE (FPS)

Breaks when significance is lost during a floating-point operation in an instruction in the specified address range. The P register points to the instruction following the one that caused the loss of significance. This event will not occur unless your program sets the floating-point loss-of-significance bit in the user mask register.

INVALID_BDP_DATA (IBD)

Breaks when a business data processing (BDP) instruction fault occurs in an instruction in the specified address range. The P register points to the instruction that caused the fault. The BDP instructions are described in the Virtual State Hardware reference manual.

READ (R)

Breaks before a read occurs from the specified address range. The break occurs only if the first byte of the item to be read is within the address range.

READ_NEXT_INSTRUCTION (RNI)

Breaks before the instruction in the specified address range is executed.

WRITE (W)

Breaks before a write occurs into the specified address range. The break occurs only if the first byte of the item to be written is within the address range.

The default event is EXECUTION.

NOTE

The following optional parameters (up to the COMMAND parameter) specify the location at which the break occurs. For the break to occur, the specified event must occur within the range defined by the address parameters. If all of these parameters are omitted, an address range of one byte is used.

LINE or *L*

Line number in the module. The module is specified by the MODULE parameter.

STATEMENT or *S*

Statement in the multi-statement line specified by the LINE parameter. The statements are numbered in consecutive order beginning with 1.

If STATEMENT is omitted, the default is 1.

STATEMENT_LABEL or *SL*

Source statement label at which to set the break. The module is specified by the MODULE parameter. The procedure is specified by the PROCEDURE parameter.

The parameter value depends on the programming language.

For FORTRAN, BASIC, and PASCAL, a statement label is an integer.

For CYBIL, a statement label is a name enclosed in slashes (/name/).

For COBOL, a statement label is a COBOL_PARAGRAPH or COBOL_SECTION identifier.

NAME or *N*

Variable name on which a READ or WRITE break is set. You must also specify EVENT=READ or EVENT=WRITE. If NAME is omitted, the break is specified by another parameter.

SECTION or *SEC*

Memory section.

$BINDING

Section containing the links to external procedures and the data of the module.

$BLANK

Section containing unnamed common.

$LITERAL

Section containing the literal data (for example, long constants) of the module.

$STATIC

Section containing the static (not on the run-time stack) variables not explicitly allocated to a named section of the module.

Unless the MODULE parameter is also specified, the section must exist for the current default module.

The SECTION parameter cannot be specified for modules that are components of a bound module unless the section is a common block.

MODULE or *M*

Module name. The module may qualify another address parameter. Otherwise it specifies the first byte of the code section of the module.

If this parameter is omitted, the current default module is used. The default module can be specified by a CHANGE_DEFAULTS command. If not specified, it is the module executing when Debug gained control.

PROCEDURE or *P*

Procedure name specifies the first byte of the code section of the procedure. Unless the MODULE parameter is specified, the procedure must exist in the current default module.

You cannot specify the LINE or SECTION address parameters with PROCEDURE.

ENTRY_POINT or *EP*

Entry point name.

You can use the BYTE_OFFSET and BYTE_COUNT parameters to modify the ENTRY_POINT parameter. You cannot use other address parameters with this parameter.

ADDRESS or *A*

Address of the first byte of memory to be changed.

Its format is rsssoooooooo(16) where r is the ring number, sss is the segment number, and oooooooo is the offset from the beginning of the segment. You can get machine addresses from the cross-reference and load maps for your program.

BYTE_OFFSET or *BO*

Offset to the base address established by one of the address parameters. The default offset is zero.

The address generated by adding BYTE_OFFSET to the base address must be within the memory block implied by the base address. The block size is the length of the section when the SECTION parameter is specified, and the length of the segment containing the machine address when the ADDRESS parameter is specified.

BYTE_COUNT or *BC*

Number of bytes in the item. The default byte count is 1.

COMMANDS or *COMMAND* or *C*

Optional string of commands to be executed by Debug, SCL, or any other active command processor when the break is honored. After the commands in the string have been executed, commands are read from the current Debug input file unless the string contains a RUN command.

If a command in the string includes a quoted string, that string must be enclosed in two single apostrophes.

No break report message is issued before the commands in the string are executed. If you want a message to be displayed, include an SCL DISPLAY_VALUE command in the string.

If an error is detected in one of the commands in the string, the break report message is issued, the error is reported, and commands are read from the Debug input file. The remaining commands in the string are not executed.

Remarks

- You specify one or more events and the location at which Debug takes control. When a specified event occurs, program execution is suspended and a message informs you which break occurred. At this point, you can enter another Debug command that can be processed by the operating system command or other active command utility (such as an SCL command).

- Debug gains control when the following events occur, even if you do not set a break for them:

  ARITHMETIC_OVERFLOW
  ARITHMETIC_SIGNIFICANCE
  DIVIDE_FAULT
  EXPONENT_OVERFLOW
  EXPONENT_UNDERFLOW
  FLOATING_POINT_INDEFINITE
  FLOATING_POINT_SIGNIFICANCE
  INVALID_BDP_DATA

  Specific breaks can be set for these events so that the specified command string can be executed when Debug gains control.

- For more information, see the Debug for NOS/VE Usage manual.

Examples Cause a break to occur when execution reaches line 10 of module PROG1:

```
DB/set_break line=10 module=prog1
--  Break name DBB$1 assigned to this break
```

Cause a break when a branch or return occurs to line 40 (of the module executing when Debug gained control):

```
DB/set_break break=b2 event=branch line=40
```

SET_SCREEN_OPTIONS DEBUG Subcommand

Purpose Enables you to change the appearance of your screen for a screen mode Debug session.

Format

SET_SCREEN_OPTIONS or
SETSO
MENU_ROWS=integer
COLUMNS=integer
SPLIT_SIZES=list of integer
STATUS=status variable

Parameters *MENU_ROWS* or *MENU_ROW* or *MR*

Specifies the number of rows of function key prompts to display on your screen. Options are:

Omitted

The number of rows of function key prompts remains the same. The default number of rows is one.

0

Displays no function key prompts.

1

Displays one row of function key prompts (functions 1 through 8).

2

Displays two rows of function key prompts (functions 1 through 16).

COLUMNS or *C*

Specifies the number of columns to be displayed for terminals that support multiple screen sizes. Options are:

Omitted

The number of columns displayed remains the same.

Integer

Specifies the number of columns to be displayed. Values can range from 40 to the maximum number allowed for your terminal screen (up to 256). The number you enter is compared to the screen sizes set up in the terminal definition for your terminal. The number of columns displayed is the closest number as large or larger than the number you enter on the COLUMNS parameter. When first entering Debug, it assumes a value of 80 columns.

SPLIT_SIZES or *SPLIT_SIZE* or *SS*

Specifies the number of lines displayed in the Source and Output windows. Options are:

Omitted

The number of lines displayed in the Source and Output windows remains the same.

List of integer

Specifies the number of lines displayed in the Source and Output windows. Values can be a list of at most two integers in the range 1 to the maximum number allowed for your terminal screen (up to 255). If two values are specified, they must be enclosed in parentheses and separated by commas or spaces. The first value specifies the number of lines displayed in the Source window and the second value specifies the number of lines displayed in the Output window (not including header information). Each window must contain at least one line.

Because the Source window is allocated first with the remainder of the screen allocated to the Output window, the second value is not necessary.

When first entering Debug, the source window occupies the top three-fourths of the screen and the Output window occupies the bottom one-fourth of the screen. The number of lines displayed is determined by the size of your terminal screen.

Remarks

- SET_SCREEN_OPTIONS, when entered in line mode, determines the screen characteristics to be displayed when screen mode Debug is activated with the ACTIVATE_SCREEN command.
- When SET_SCREEN_OPTIONS is entered on the home line while in screen mode, the screen is updated immediately according to the parameters specified.
- For all omitted parameters, Debug assumes you want the same value used the last time you entered the SET_SCREEN_OPTIONS command in the current Debug session. If SET_SCREEN_OPTIONS has not been entered, Debug assumes the default values.
- Screen characteristics remain in effect throughout the Debug session until they are changed by another SET_SCREEN_OPTIONS command or by the tailoring functions of screen mode Debug. When you end the Debug session, all screen characteristics return to their default values.
- You can also include the SET_SCREEN_OPTIONS command in the debug_input file so that your screen characteristics are modified immediately when you begin a Debug session.
- For more information, see the Debug for NOS/VE Usage manual.

Examples

The following example displays (when screen mode is activated) two rows of function key prompts, 132 columns, and a source window as large as possible.

```
DB/set_screen_options menu_rows=2 columm=132 split_size=255
```

SET_STEP_MODE DEBUG Subcommand

Purpose Activates or deactives step mode. In step mode, control is returned after a specified subset of a task is executed.

Format

SET_STEP_MODE or
SETSM
MODE=keyword
UNIT=keyword
MODULE=list of module or *keyword*
PROCEDURE=list of procedure or *keyword*
SPAN=integer
COMMAND=string
STATUS=status variable

Parameters

MODE

Indicates whether to activate or deactivate step mode. This parameter is required.

ON

Activates step mode.

OFF

Deactivates step mode. When step mode is off, any remaining parameters are ignored.

If you specify MODE=ON and step mode is already on, all previous values are replaced with the new parameter values.

UNIT or *U*

Length of each step. The default value is LINE.

LINE (L)

The step is reported before the code is executed for each line, except for the procedure lines.

PROCEDURE (P)

The step is reported each time a new procedure begins and after any prolog code for the procedure has executed.

COBOL_SECTION (CS)

The step is reported each time a section header is reached (COBOL programs only).

COBOL_PARAGRAPH (CP)

The step is reported each time a paragraph is reached (COBOL programs only).

MODULE or *M*

Used with the UNIT parameter to specify the modules reported. If this parameter is omitted, the current default module is used.

$ALL

A step is reported that is in any module.

$CURRENT

A step is reported only if the step occurs in the module where the program is executing when step mode is activated.

list of names

A step is reported if the step occurs in any of the named modules.

You cannot specify both the MODULE and PROCEDURE parameters in the same SET_STEP_MODE command.

PROCEDURE or *P*

Used with the UNIT parameter to specify the procedure reported. If the parameter is omitted, the current default procedure is used.

$ALL

A step is reported that is in any procedure.

$CURRENT

A step is reported only if the step occurs in the procedure where the program is executing when step mode is activated.

list of names

A step is reported if the step occurs in any of the named procedures.

You cannot specify both the MODULE and PROCEDURE parameters in the same SET_STEP_MODE command.

SPAN or *S*

Number of steps to occur before execution stops and the step is reported. By default, every step that occurs is reported.

COMMAND or *COMMANDS* or *C*

Optional string of commands to be executed when the step occurs.

If the string of commands includes a RUN command, the task is resumed and the step is not reported.

If the string does not include a RUN command, command input will be requested from the current DEBUG_INPUT file after the string of commands has been executed.

Remarks

- If step mode is activated, a RUN command causes your program to execute for the specified unit. You are then prompted for further command input.
- A string of commands can be associated with the step and will be processed each time the step is completed. Stepping with a unit of line or procedure is only available if the source program was compiled with OPT=DEBUG.
- Activating step mode is an effective debugging aid, but it uses a lot of execution time.
- For more information, see the Debug for NOS/VE Usage manual.

Examples The following command sequence shows a command to turn on step mode, two RUN commands to execute two steps, and a command to turn off step mode. The value of variable x is displayed at each step.

```
DB/set_step_mode,on,command='display_program_value,x'
DB/run
x =   2.00000000000000E+0000
 --  DEBUG: step at M=$MAIN L=34 BO=212
DB/run
x =   2.00000000000000E+0000
 --  DEBUG: step at M=$MAIN L=35 BO=6
DB/set_step_mode off
```

EDIT_CATALOG 13

EDIT_CATALOG Command

Purpose Accesses the EDIT_CATALOG (EDIC) utility, a full screen application that can be used to create, move, copy, print, view, edit, and execute files.

Format

EDIT_CATALOG or
EDIC
CATALOG=file
DISPLAY_OPTIONS=keyword
NO_DOLLAR_FILES=boolean
STATUS=status variable

Parameters

CATALOG or *C*

Catalog to be displayed. Omission causes the system to display the current working catalog.

DISPLAY_OPTIONS or *DISPLAY_OPTION* or *DO*

File information to be displayed.

ALL (A)

All file attributes are displayed.

BRIEF (B)

Only the name and entry type (file or catalog) are displayed.

The default is BRIEF.

NO_DOLLAR_FILES or *NDF*

Boolean indicating whether file names containing a dollar sign are to be omitted from the display. (By convention, a dollar sign character [$] appears only in CDC-defined file names.)

TRUE (ON or YES)

File names containing a $ character are not displayed.

FALSE (OFF or NO)

File names containing a $ character are displayed.

The default is FALSE.

Remarks For more information, see the NOS/VE System Usage manual.

$CURRENT_FILE
EDIC Function

Purpose Specifies the current file. This function can be used instead of explicitly naming a file within an SCL command you enter from within EDIT_CATALOG. The current file is considered to be the file at which the cursor was positioned before you pressed HOME. If you use this function within an SCL command and did not previously position the cursor on a file name, an error occurs.

Format **$CURRENT_FILE** or
$CF

Parameters None.

Remarks

- Evaluation of the $CURRENT_FILE function must occur within EDIT_CATALOG. Consequently, if you enter a command which initiates the execution of another task and specify the $CURRENT_FILE as a parameter, the $CURRENT_FILE will not be evaluated and you will receive an error message regarding the filename $CURRENT_FILE.

  If you execute a procedure and spe cify $CURRENT_FILE as one of the parameters, the $CURRENT_FILE will be evaluated.

- For more information, see the NOS/VE System Usage manual.

SET_DISPLAY_OPTION
EDIC Subcommand

Purpose Specifies the display option you wish to see while using the EDIT_CATALOG command.

Format

SET_DISPLAY_OPTION or
SET_DISPLAY_OPTIONS or
SETDO
DISPLAY_OPTIONS=keyword
STATUS=status variable

Parameters *DISPLAY_OPTIONS* or *DISPLAY_OPTION* or *DO*

Specifies the amount of information that you want to have displayed. The following are possible entries:

BRIEF (B)

Selects a display in which file and catalog names are displayed. The display also includes the notation (catalog) indicating that a displayed name is a catalog.

ALL (A)

Selects a display showing all file and catalog information.

If the DISPLAY_OPTION parameter is omitted, BRIEF is used.

Remarks For more information, see the NOS/VE System Usage manual.

SET_SCREEN_OPTION
EDIC Subcommand

Purpose Specifies the number of rows of keys to be displayed at the bottom of your screen with the SET_SCREEN_OPTION subcommand.

Format

SET_SCREEN_OPTION or
SET_SCREEN_OPTIONS or
SETSO
MENU_ROWS=integer
STATUS=status variable

Parameters *MENU_ROWS* or *MR*

Specifies the number of rows of keys to be displayed. You may specify a value of zero, one, or two. If omitted, a value of one is assumed.

Remarks For more information, see the NOS/VE System Usage manual.

EDIT_DECK 14

EDIT_DECK
EDID Subcommand

Purpose Opens the specified deck in the working library for editing while maintaining your current position in other decks.

Format

EDIT_DECK or
EDID
DECK=name
STATUS=status variable

Parameters

DECK or D

Specifies the deck to be edited. If the deck does not exist, it is created.

This parameter is required.

Remarks

- To discard decks created unintentionally, enter:

```
end_deck write_deck=false
```

- For more information, see the NOS/VE File Editor manual.

EDIT_FIRST_DECK
EDID Subcommand

Purpose Opens the first deck on the working library for editing while maintaining your current position in other decks.

Format

EDIT_FIRST_DECK or
EDIFD
STATUS=status variable

Remarks

- Decks are always in alphabetical order in the working library.
- For more information, see the NOS/VE File Editor manual.

EDIT_LAST_DECK
EDID Subcommand

Purpose Opens the last deck in the working library for editing while maintaining your current position in other decks.

Format

EDIT_LAST_DECK or
EDILD
STATUS=status variable

Remarks

- Decks are always in alphabetical order in the working library.
- For more information, see the NOS/VE File Editor manual.

EDIT_NEXT_DECK
EDID Subcommand

Purpose Opens the next deck on the working library for editing while maintaining your current position in other decks.

Format

EDIT_NEXT_DECK or
EDIND
STATUS=status variable

Remarks

- Decks are always in alphabetical order in the working library.
- For more information, see the NOS/VE File Editor manual.

END_DECK
EDID Subcommand

Purpose Closes editing on the current deck.

Format

END_DECK or
ENDD
WRITE_DECK=boolean
STATUS=status variable

Parameters *WRITE_DECK* or *WD* or *WRITE_FILE* or *WF*

Specifies whether the changes made to the deck since it was opened for editing are to be written to the working library.

TRUE indicates that the deck is to be rewritten.

FALSE indicates that the deck remains unchanged (the edited copy is discarded). FALSE also discards a deck that has been created during the current editing session provided that you have not closed the deck. This is the easiest way to delete decks that were unintentionally created.

If omitted, TRUE is assumed and the results are written to the working library.

Remarks For more information, see the NOS/VE File Editor manual.

RESET_DECK
EDID Subcommand

Purpose Discards changes made to the current deck being edited. All changes made since the last time the deck was opened for editing are discarded. The editor obtains a new copy of the deck from the working library.

Format

RESET_DECK or
RESD
STATUS=status variable

Remarks For more information, see the NOS/VE File Editor manual.

SELECT_DECK
EDID Subcommand

Purpose Opens the specified deck on the working library for editing and closes the previous deck (if any).

Format

SELECT_DECK or
SELD
DECK=name
STATUS=status variable

Parameters **DECK or D**

Specifies the name of the deck to be edited. If the deck does not exist, it is created.

This parameter is required.

Remarks

- Decks are always in alphabetical order in the working library.
- For more information, see the NOS/VE File Editor manual.

SELECT_FIRST_DECK EDID Subcommand

Purpose Opens the first deck on the working library for editing and closes the previous deck (if any).

Format

SELECT_FIRST_DECK or
SELFD
STATUS=status variable

Remarks

- Decks are always in alphabetical order in the working library.
- For more information, see the NOS/VE File Editor manual.

SELECT_LAST_DECK EDID Subcommand

Purpose Opens the last deck on the working library for editing and closes the previous deck (if any).

Format

SELECT_LAST_DECK or
SELLD
STATUS=status variable

Remarks

- Decks are always in alphabetical order in the working library.
- For more information, see the NOS/VE File Editor manual.

SELECT_NEXT_DECK
EDID Subcommand

Purpose Opens the next deck on the working library for editing and closes the previous deck (if any).

Format

SELECT_NEXT_DECK or
SELND
STATUS=status variable

Remarks

- Decks are always in alphabetical order in the working library.
- For more information, see the NOS/VE File Editor manual.

EDIT_FILE 15

EDIT_FILE Command

Purpose Starts a file editor (EDIT_FILE utility) session.

Format

EDIT_FILE or
EDIF
FILE=file
INPUT=file
OUTPUT=file
PROLOG=file
DISPLAY_UNPRINTABLE_CHARACTERS=boolean
STATUS=status variable

Parameters

FILE or **F**

Specifies the name of the file you want to edit. If the file you specify does not exist, a new file is created.

The file cannot be an object file.

This parameter is required.

INPUT or *I*

Specifies the file to be used as input to the editor. This file can be positioned. This file contains optional editor subcommands used to manipulate the working file. If omitted, $COMMAND is assumed.

OUTPUT or *O*

Specifies the file to which you want to write any output that may result from your editing session. This file can be positioned.

If OUTPUT is omitted, $OUTPUT is assumed. File $OUTPUT is usually connected to the terminal.

PROLOG or *P*

Specifies the file containing subcommands you want executed each time you start the editor.

If omitted, $USER.SCU_EDITOR_PROLOG is assumed.

DISPLAY_UNPRINTABLE_CHARACTERS or *DUC*

Specifies whether unprintable ASCII characters are replaced by mnemonics when the file is displayed at the terminal. Options are:

TRUE

Unprintable characters (ASCII values 127 and 0 through 31) are replaced by their respective mnemonic values enclosed within the less than and greater than characters, < >. The mnemonics are replaced by the ASCII characters when the file is replaced.

FALSE

Unprintable characters are replaced by a single space and a warning message is issued. If the file is written when you exit the editing session, the unprintable characters are replaced by spaces.

If DISPLAY_UNPRINTABLE_CHARACTERS is omitted, FALSE is used.

ASCII characters and their corresponding mnemonic values are listed in appendix C.

Remarks

- If you would like to specify a file containing editor subcommands to be executed when you leave the editor (an epilog file), use the SET_EPILOG subcommand. If you want this done each time, include SETE in the file you specify for the PROLOG parameter.
- The following prompt appears for line editing:

  ```
  ef/
  ```

- To edit a second file while in the editor, enter the EDIT_FILE subcommand. The FILE and STATUS parameters are the only parameters allowed on the EDIT_FILE subcommand.
- For more information, see the NOS/VE File Editor manual.

Examples

The following command starts the EDIT_FILE utility with file $USER.MY_FILE:

```
edit_file file=$user.my_file
```

ACTIVATE_SCREEN EDIF Subcommand

Purpose Activates screen mode; specifies terminal type.

Format

ACTIVATE_SCREEN or
ACTS
MODEL=name
STATUS=status variable

Parameters *MODEL* or *M*

Specifies the type of terminal you are using. Valid entries are:

Entry	Terminal
CDC_721	Control Data 721
CDC_722	Control Data 722
CDC_722_30	Control Data 722-30
MAC_CONNECT_10	Apple Macintosh running version 1.0 or 1.0+ of Control Data CONNECT for the Macintosh
MAC_CONNECT_11	Apple Macintosh running version 1.1 of Control Data CONNECT for the Macintosh
PC_CONNECT_10	IBM PC or equivalent running version 1.0 of Control Data CONNECT for the IBM PC
PC_CONNECT_11	IBM PC or equivalent running version 1.1 of Control Data CONNECT for the IBM PC
PC_CONNECT_12	IBM PC or equivalent running version 1.2 of Control Data CONNECT for the IBM PC
PC_CONNECT_13	IBM PC or equivalent running version 1.3 of Control Data CONNECT for the IBM PC

DEC_VT100_GOLD	Digital Equipment VT100
DEC_VT220	Digital Equipment VT220
ZEN_Z19	Zenith Z19 or Heathkit H19
ZEN_Z29	Zenith Z29

If the terminal you are using is not on this list, ask site personnel for the entry that activates your screen.

If the MODEL parameter was not specified on an earlier ACTIVATE_SCREEN or SET_SCREEN_OPTIONS subcommand or on the TERMINAL_MODEL parameter of the CHANGE_TERMINAL_ATTRIBUTES command, it is required.

Remarks

- The recommended method for preparing your session for screen editing is to enter the CHANGE_TERMINAL_ATTRIBUTES and CHANGE_INTERACTION_STYLE commands, described in the NOS/VE System Usage manual, before you start an editing session. To do this automatically, include these commands in your user prolog.
- Inside procedures, you can use the ACTIVATE_SCREEN subcommand to allow the user of the procedure to enter editor subcommands.
- Executing this subcommand causes the firmware of some terminals to be reinitialized. Refer to your terminal's documentation for more information.
- Use the $SCREEN_ACTIVE function to determine whether screen mode is active.
- For more information, see the NOS/VE File Editor manual.

Examples

To switch from line mode to screen mode in an editing session, enter:

```
activate_screen
```

$ACTIVE_IDENTIFIER
EDIF Function

Purpose Returns a line identifier string (for editing decks only) that indicates if the line you specify is active.

Format

$ACTIVE_IDENTIFIER or
$AI
(lines)

Parameters

lines

Identifies the line for which you want to find the status. If the line you specify is active, the same string is returned. If the line is not active, the line identifier for the nearest active line is returned. If no lines are active, FIRST is returned.

This parameter is required.

Remarks For more information, see the NOS/VE File Editor manual.

ALIGN_SCREEN
EDIF Subcommand

Purpose Enables you to change the alignment of your screen.

Format

ALIGN_SCREEN or
ALIS or
A
MIDDLE=lines or *keyword*
TOP=lines or *keyword*
BOTTOM=lines or *keyword*
OFFSET=integer
STATUS=status variable

Parameters

MIDDLE or *M*

Specifies a line to be centered vertically on the screen. Values can be an integer, line identifier, or one of the keywords: CURRENT, FIRST, FIRST_MARK, FIRST_SCREEN, LAST, LAST_MARK, LAST_SCREEN. You cannot use this parameter with the TOP and BOTTOM parameters.

If you omit this parameter, CURRENT is assumed.

TOP or *T*

Specifies a line to be positioned at the top of the screen. The resulting middle line of the screen becomes the current line. Values can be an integer, line identifier, or one of the keywords: CURRENT, FIRST, FIRST_MARK, FIRST_SCREEN, LAST, LAST_MARK, LAST_SCREEN. You cannot use this parameter with the MIDDLE and BOTTOM parameters.

If you omit this parameter, no value is supplied.

BOTTOM or *B*

Specifies a line to appear at the bottom of the screen. The resulting middle line of the screen becomes the current line. Values can be an integer, line identifier, or one of the keywords: CURRENT, FIRST, FIRST_MARK, FIRST_SCREEN, LAST, LAST_MARK, LAST_SCREEN. You cannot use this parameter with the TOP and MIDDLE parameters.

If you omit this parameter, no value is supplied.

OFFSET or *O*

Specifies the number of columns to offset your view of the file on the screen. The number can be an integer from 0 through 216. The number you specify is added to column 1 and the last column displayed. For example, if the rightmost column is 80 and you specify an OFFSET value of 20, the leftmost column becomes 21 and the rightmost column becomes 100.

Remarks

- You can use $OFFSET to return the current OFFSET value.
- For more information, see the NOS/VE File Editor manual.

Examples

- The following example moves the current line to the bottom of the screen (same as the LinDn operation):

```
align_screen bottom=current
```

- The following example displays column 51 as the leftmost column:

```
alis offset=50
```

BREAK_TEXT
EDIF Subcommand

Purpose Breaks a line at a specific point in the line to make one line into two lines.

Format

BREAK_TEXT or
BRET or
B

LINES=lines or *keyword*
COLUMN=integer or *keyword*
STATUS=status variable

Parameters *LINES* or *LINE* or *L*

Identifies the line to be broken. Values can be an integer, line identifier, or one of the keywords: CURRENT, FIRST, FIRST_MARK, FIRST_SCREEN, LAST, LAST_MARK, LAST_SCREEN. Ranges are not allowed.

If omitted, CURRENT is assumed.

COLUMN or *C*

Specifies the column before which the break is to occur. In other words, the break occurs just before the column specified. Values can be an integer from 1 through 256, or one of the keywords: CURRENT, FIRST_MARK, LAST_MARK, MAXIMUM.

If omitted, CURRENT is assumed.

Remarks For more information, see the NOS/VE File Editor manual.

CENTER_LINES
EDIF Subcommand

Purpose Centers a line or lines between margins set with the SET_PARAGRAPH_MARGINS subcommand.

Format

CENTER_LINES or
CENTER_LINE or
CENL

NUMBER=integer or *keyword*
LINES=range of lines or *keyword*
STATUS=status variable

Parameters *NUMBER* or *N*

Specifies the number of lines to be centered.

If you omit this parameter and specify a range for the LINE parameter, NUMBER assumes a value of ALL.

If you omit this parameter without specifying a range of lines, NUMBER assumes a value of 1.

If NUMBER and LINES are both omitted, CURRENT is assumed.

LINES or *LINE* or *L*

Specifies a range of lines to be centered.

If one line is specified, the centering is limited to that line. Values can be an integer, line identifier, or one of the keywords: ALL, CURRENT, FIRST, FIRST_MARK, FIRST_SCREEN, LAST, LAST_MARK, LAST_SCREEN. MARK, SCREEN.

If LINES is omitted, the lines to be centered are determined by the NUMBER parameter. If LINES and NUMBER are both omitted, CURRENT is assumed.

Remarks For more information, see the NOS/VE File Editor manual.

Examples

- The following example centers the next five lines.

```
center_lines number=5
```

- The following example centers all lines between lines 15 and 23.

```
cenl line=15..23
```

CLEAR_TABS
EDIF Subcommand

Purpose Deletes all or some of the tab columns.

Format

CLEAR_TABS or
CLEAR_TAB or
CLET
TAB_COLUMN=list of range of integer or *keyword*
STATUS=status variable

Parameters *TAB_COLUMN* or *TAB_COLUMNS* or *TC*

Specifies the columns to delete as tab columns. Values can be the keyword ALL or a list of a range of integers from 1 through 256.

If TAB_COLUMN is omitted, all tabs are cleared.

Remarks For more information, see the NOS/VE File Editor manual.

Examples The following CLEAR_TAB subcommand clears columns 7 and 65 as tab columns:

```
clear_tab tab_column=(7,65)
```

COPY_TEXT
EDIF Subcommand

Purpose Copies a block of text from one place to another within your working files.

Format

COPY_TEXT or
COPT or
C

TEXT=range of string
NUMBER=integer or *keyword*
LINES=range of lines or *keyword*
COLUMNS=range of integer or *keyword*
INSERTION_LOCATION=lines or *keyword*
INSERTION_COLUMN=integer or *keyword*
PLACEMENT=keyword
BOUNDARY=keyword
UPPER_CASE=boolean
WORD=boolean
REPEAT_SEARCH=boolean
STATUS=status variable

Parameters *TEXT* or *T*

Specifies strings of text in the first and last lines of a block of text to be copied. If you enter only one string, the block of text to be copied will contain only one line. If you enter two strings, the search for the second begins immediately after the first is found.

If TEXT is specified, the INSERTION_COLUMN and BOUNDARY parameters are ignored and line boundaries are used.

If omitted, the lines to be copied will be determined by the NUMBER and LINES parameters or the REPEAT_SEARCH parameter.

NUMBER or *N*

Specifies the number of blocks of text to be copied. Values for this parameter can be numbers or the keyword ALL (A).

If omitted and a range is specified for the LINES parameter, NUMBER assumes a value of ALL. Otherwise the assumed value is 1.

LINES or *LINE* or *L*

Specifies the range of lines to be searched for the text to be copied. If a single value is specified, only that line is searched. Values can be an integer, line identifier, or one of the keywords: ALL, CURRENT, FIRST, FIRST_MARK, FIRST_SCREEN, LAST, LAST_MARK, LAST_SCREEN, MARK, SCREEN.

If omitted, CURRENT..LAST is assumed.

COLUMNS or *COLUMN* or *C*

Specifies the range of columns to be searched for text to be copied. The integers can be from 1 through 256 or any of the keywords: CURRENT, FIRST_MARK, LAST_MARK, MARK, MAXIMUM.

If omitted, CURRENT is assumed. If omitted and you have specified LINE=MARK, the marked lines provide the column boundaries. If COLUMN is omitted and you have specified a LINE parameter other than MARK, all columns will be searched.

INSERTION_LOCATION or *IL*

Specifies the line before or after which the text is to be copied (depending on the value of the PLACEMENT parameter). Values can be an integer, line identifier, or one of the LINE keywords: CURRENT, FIRST, FIRST_MARK, FIRST_SCREEN, LAST, LAST_MARK, LAST_SCREEN. Ranges are not allowed.

If omitted, CURRENT is assumed.

INSERTION_COLUMN or *IC*

Specifies the column before or after which the text is to be copied (depending on the value of the PLACEMENT parameter). Values can be an integer from 1 through 256, or any of the COLUMN keywords: CURRENT, FIRST_MARK, LAST_MARK, MAXIMUM. Ranges are not allowed.

If omitted, CURRENT is assumed.

If a value for TEXT is specified, INSERTION_COLUMN is ignored.

PLACEMENT or *P*

Specifies if the copied lines are to appear BEFORE (B) or AFTER (A) the location specified by the INSERTION_LOCATION parameter.

If omitted, AFTER is assumed.

BOUNDARY or *B*

Specifies the type of boundary that will limit the search. Values can be LINE or STREAM.

If BOUNDARY and COLUMNS are both omitted, LINE is assumed.

If BOUNDARY is omitted but COLUMNS is specified, STREAM is assumed.

If a value for TEXT is specified, BOUNDARY is ignored; line boundaries are used.

UPPER_CASE or *UC*

Determines the significance of capitalization in the search.

If you specify TRUE, the editor searches the file as if it were all uppercase.

If you specify FALSE, the editor searches for the text exactly as it was entered.

If omitted, FALSE is assumed unless you specify TRUE for REPEAT_SEARCH. In this case, your last value for UPPER_CASE is used.

WORD or *W*

Determines whether the editor searches for the specified text string as a word (the text you want to search for is surrounded by nonalphanumeric characters).

If you specify TRUE, the editor searches for the text as a word. If you specify FALSE, it doesn't.

If omitted, FALSE is assumed unless you specify TRUE for REPEAT_SEARCH. In this case, your last value for WORD is used.

REPEAT_SEARCH or *RS*

Instructs the editor how to use the values entered for the last TEXT, UPPER_CASE, and WORD parameters.

If you specify TRUE, the editor uses the same TEXT, UPPER_CASE, and WORD parameters as the last time you entered them for any subcommand (unless you have specified values for this subcommand).

If you specify FALSE, the editor uses the parameters entered with the current subcommand.

If omitted, FALSE is assumed.

Remarks For more information, see the NOS/VE File Editor manual.

Examples

- The following copies lines 30 through 40 to immediately after the current line:

```
copy_text line=30..40
```

- The following copies the next occurrence of a block of text beginning with the line containing *one* and ending with the line containing *five* to immediately before line 71:

```
copt text='one'..'five' insertion_location=71 ..
    placement=backward
```

$CURRENT_COLUMN
EDIF Function

Purpose Returns the value of the current column number.

Format **$CURRENT_COLUMN** or
$CC

Parameters None.

Remarks

- If the POSITION_CURSOR subcommand is used to specify a column on a row that is not part of the file text, the value returned is the column at which the cursor was positioned before the POSITION_CURSOR subcommand was entered.
- For more information, see the NOS/VE File Editor manual.

$CURRENT_DECK
EDIF Function

Purpose Returns a string specifying the current deck's name (for editing decks only).

Format **$CURRENT_DECK** or
$CD

Parameters None.

Remarks

- You can also use the $CURRENT_OBJECT function for this purpose.
- All letters in the string are uppercase, even if the name was originally entered using lowercase letters.
- For more information, see the NOS/VE File Editor manual.

$CURRENT_LINE
EDIF Function

Purpose Returns an integer specifying the current line number.

Format **$CURRENT_LINE** or
$CL

Parameters None.

Remarks For more information, see the NOS/VE File Editor manual.

Examples The following statements assign variable LAST_LINE an integer value reflecting the number of lines in a file or deck:

```
position_cursor l=last
last_line = $current_line
```

$CURRENT_OBJECT
EDIF Function

Purpose Returns a string identifying the current file name or deck name.

Format **$CURRENT_OBJECT** or
$CO

Parameters None.

Remarks

- You can use the $CURRENT_OBJECT_TYPE function to determine if the string is a file name or a deck name.
- For more information, see the NOS/VE File Editor manual.

Examples The following procedure rewrites the current file, or if you are editing a deck or a local file, the procedure places a copy of the deck or file in the catalog $USER.SAVE_EDITOR_FILES. This catalog must be present in your $USER catalog.

```
PROC checkpoint_file, chef (status)

IF $file($fname($co), permanent) THEN
  write_file f=$fname($co)
ELSE
  write_file f=$fname('$user.save_editor_files.'//..
$path($fname($co), last))
IFEND

PROCEND checkpoint_file
```

$CURRENT_OBJECT_TYPE
EDIF Function

Purpose Returns a string identifying the current object being edited. Possible values are FILE, DECK, or NULL.

Format **$CURRENT_OBJECT_TYPE** or
$COT

Parameters None.

Remarks For more information, see the NOS/VE File Editor manual.

$CURRENT_ROW
EDIF Function

Purpose Returns an integer identifying the row on the screen where the cursor is positioned (as opposed to the current line number of a file).

Format **$CURRENT_ROW** or
$CR

Parameters None.

Remarks

- Zero is returned if the current row is not within screen boundaries or if you are in line mode.
- For more information, see the NOS/VE File Editor manual.

$CURRENT_SPLIT
EDIF Function

Purpose Returns an integer specifying the split of the screen in which the cursor is positioned.

Format **$CURRENT_SPLIT** or
$CS

Parameters None.

Remarks

- If you are in line mode, zero is returned.
- Values returned can be from 1 through 16. The top split of the screen is 1, the next lower is 2, and so on.
- For more information, see the NOS/VE File Editor manual.

$CURRENT_WORD EDIF Function

Purpose Returns the current word as a string.

Format **$CURRENT_WORD** or
$CW

Parameters None.

Remarks

- This function is particularly useful when supplied as the value for the TEXT parameter for the LOCATE_TEXT and REPLACE_TEXT subcommands.
- For more information, see the NOS/VE File Editor manual.

Examples

- The following example converts the characters in the current word to lower case:

```
replace_text t=$cw ..
   nt=$translate(upper_to_lower,$cw)
```

- If the current word is a deck name, you can edit that deck by entering the following command:

```
edit_deck d=$name($cw)
```

- If the current word is a command name, you can display information about that command by entering the following command:

```
include_command ..
    c='display_command_information c='//$cw
```

$CURRENT_WORD_COLUMN
EDIF Function

Purpose Returns an integer specifying the column in which the current word begins.

Format **$CURRENT_WORD_COLUMN** or
$CWC

Parameters None.

Remarks For more information, see the NOS/VE File Editor manual.

Examples The following marks the current word:

```
mark_character c=$cwc..$strlen($cw)+$cwc-1
```

DEACTIVATE_SCREEN
EDIF Subcommand

Purpose Stops screen mode without stopping the editor.

Format **DEACTIVATE_SCREEN** or
DEAS
STATUS=status variable

Remarks

- When you enter this subcommand, the screen is cleared and the line mode prompt appears:

  ```
  ef/
  ```

- Use DEACTIVATE_SCREEN to enter line mode commands that must be continued over more than one input line.

- For more information, see the NOS/VE File Editor manual.

DELETE_CHARACTERS EDIF Subcommand

Purpose Enables you to delete characters.

Format

DELETE_CHARACTERS or
DELC or
DELETE_CHARACTER or
DC

NUMBER=integer or *keyword*
LINES=lines or *keyword*
COLUMNS=range of integer or *keyword*
STATUS=status variable

Parameters

NUMBER or *N*

Specifies the number of characters to be deleted. Values may be an integer or the keyword ALL.

If you omit this parameter without specifying LINE or COLUMN, a value of 1 is assumed.

If you omit this parameter and specify a range for COLUMN, ALL is assumed.

LINES or *LINE* or *L*

Specifies the line in which characters will be deleted. Values can be an integer, line identifier, or any of the keywords: CURRENT, FIRST, FIRST_MARK, FIRST_SCREEN, LAST, LAST_MARK, LAST_SCREEN. Ranges are not allowed.

If omitted, CURRENT is assumed.

COLUMNS or *COLUMN* or *C*

Specifies the columns to be deleted within the specified line(s). Values can be an integer from 1 through 256, or one of the keywords: CURRENT, FIRST_MARK, LAST_MARK,MAXIMUM.

If omitted, CURRENT is assumed.

Remarks For more information, see the NOS/VE File Editor manual.

Examples The following deletes the characters in columns 1 through 17 of the current line.

```
delete_characters columns=(1..17)
```

DELETE_EMPTY_LINES
EDIF Subcommand

Purpose Deletes a block of blank lines until a nonblank line is encountered.

Format

DELETE_EMPTY_LINES or
DELETE_EMPTY_LINE or
DELEL
LINES=lines or *keyword*
STATUS=status variable

Parameters *LINES* or *LINE* or *L*

Specifies the line at which the deletion of blank lines is to begin. Values can be an integer, line identifier, or any of the keywords: CURRENT, FIRST, FIRST_MARK, FIRST_SCREEN, LAST, LAST_MARK, LAST_SCREEN. Ranges are not allowed.

If the line you specify is not a blank line, nothing happens.

If LINES is omitted, CURRENT is assumed.

Remarks For more information, see the NOS/VE File Editor manual.

DELETE_LINES
EDIF Subcommand

Purpose Enables you to delete a line or range of lines.

Format

DELETE_LINES or
DELETE_LINE or
DELL
TEXT=range of string
NUMBER=integer or *keyword*
LINES=range of lines or *keyword*
UPPER_CASE=boolean
WORD=boolean
REPEAT_SEARCH=boolean
STATUS=status variable

Parameters *TEXT* or *T*

Specifies a block of text to be deleted, beginning with the line containing the first string and ending with the line containing the second string.

If TEXT is omitted, the editor does not supply a value and the NUMBER and LINE parameters determine which text will be deleted.

NUMBER or *N*

Specifies the number of lines to be deleted, or the number of blocks of text to be deleted depending on the values you specify for the LINES and TEXT parameter. Values can be an integer or the keyword ALL.

If you omit this parameter and specify a range for the LINE parameter, NUMBER assumes a value of ALL.

If you omit this parameter without specifying a range of lines, NUMBER assumes a value of 1.

LINES or *LINE* or *L*

Specifies a range of lines to be deleted. Values can be an integer, line identifier, or any of the keywords: ALL, CURRENT, FIRST, FIRST_MARK, FIRST_SCREEN, LAST, LAST_MARK, LAST_SCREEN, MARK, SCREEN.

If a single integer or keyword is specified, only that line is deleted.

If LINE=MARK is specified, marked lines are deleted in their entirety (even if the boundary implied by the mark is STREAM).

If LINE is omitted, CURRENT..LAST is assumed.

UPPER_CASE or *UC*

Determines the significance of capitalization in the search.

If you specify TRUE, the editor searches the file as if it were all uppercase.

If you specify FALSE, the editor searches for the text exactly as it was entered.

If omitted, FALSE is assumed unless you specify TRUE for REPEAT_SEARCH. In this case, your last value for UPPER_CASE is used.

WORD or *W*

Determines whether the editor searches for the specified text string as a word (the text you want to search for is surrounded by nonalphanumeric characters).

If you specify TRUE, the editor searches for the text as a word. If you specify FALSE, it doesn't.

If omitted, FALSE is assumed unless you specify TRUE for REPEAT_SEARCH. In this case, your last value for WORD is used.

REPEAT_SEARCH or *RS*

Instructs the editor how to use the values entered for the last TEXT, UPPER_CASE, and WORD parameters.

If you specify TRUE, the editor uses the same TEXT, UPPER_CASE, and WORD parameters as the last time you entered them for any subcommand (unless you have specified values for this subcommand).

If you specify FALSE, the editor uses the parameters entered with the current subcommand.

If omitted, FALSE is assumed.

Remarks For more information, see the NOS/VE File Editor manual.

Examples The following subcommand deletes marked lines.

```
delete_lines line=mark
```

DELETE_TEXT
EDIF Subcommand

Purpose Enables you to delete blocks of text.

Format

DELETE_TEXT or
DELT or
D

TEXT=range of string
NUMBER=integer or *keyword*
LINES=range of lines or *keyword*
COLUMNS=range of integer or *keyword*

BOUNDARY=keyword
UPPER_CASE=boolean
WORD=boolean
REPEAT_SEARCH=boolean
STATUS=status variable

Parameters *TEXT* or *T*

Specifies strings of text in the first and last lines of a block of text to be deleted. If you enter only one string, the block of text to be deleted will contain only one line. If you enter two strings, the search for the second begins immediately after the first is found.

If omitted, the lines to be deleted will be determined by the NUMBER and LINES parameters or the REPEAT_SEARCH parameter.

NUMBER or *N*

Specifies the number of lines to be deleted. Values can be numbers or the keyword ALL.

If you omit this parameter and specify a range for the LINE parameter, NUMBER assumes a value of ALL.

If you omit this parameter without specifying a range of lines, NUMBER assumes a value of 1.

LINES or *LINE* or *L*

Specifies a range of lines to be deleted.

If a single value is specified, only that line is deleted.

If omitted, CURRENT..LAST is assumed.

COLUMNS or *COLUMN* or *C*

Specifies the columns to be deleted in the specified lines.

If specified, deletion occurs from the first line and column to the last line and column.

If omitted, the entire line is deleted.

BOUNDARY or *B*

Specifies the type of boundary that will limit the search. Values can be LINE or STREAM.

If BOUNDARY is omitted, LINE is assumed.

If COLUMN is specified, STREAM is assumed.

UPPER_CASE or *UC*

Determines the significance of capitalization in the search.

If you specify TRUE, the editor searches the file as if it were all uppercase.

If you specify FALSE, the editor searches for the text exactly as it was entered.

If omitted, FALSE is assumed unless you specify TRUE for REPEAT_SEARCH. In this case, your last value for UPPER_CASE is used.

WORD or *W*

Determines whether the editor searches for the specified text string as a word (the text you want to search for is surrounded by nonalphanumeric characters).

If you specify TRUE, the editor searches for the text as a word. If you specify FALSE, it doesn't.

If omitted, FALSE is assumed unless you specify TRUE for REPEAT_SEARCH. In this case, your last value for WORD is used.

REPEAT_SEARCH or *RS*

Instructs the editor on how to use the values entered for the last TEXT, UPPER_CASE, and WORD parameters.

TRUE instructs the editor to use the same TEXT, UPPER_CASE, and WORD parameters as the last time you entered them on any subcommand, unless you have specified values for them on this subcommand.

FALSE instructs the editor to use the parameters entered with the current DELETE_TEXT subcommand.

If omitted, FALSE is assumed.

Remarks For more information, see the NOS/VE File Editor manual.

Examples The following deletes all lines from the line containing first to the line containing last.

```
delete_text text='first'..'last'
```

DELETE_WORD
EDIF Subcommand

Purpose Deletes words, blanks, or characters, depending on the position in the file you specify.

Format

DELETE_WORD or
DELW or
DW
LINES=lines or *keyword*
COLUMN=integer or *keyword*
STATUS=status variable

Parameters

LINES or *LINE* or *L*

Specifies a line in which the deletion is to occur. Values can be an integer, line identifier, or any of the keywords: CURRENT, FIRST, FIRST_MARK, FIRST_SCREEN, LAST, LAST_MARK, LAST_SCREEN. Ranges are not allowed.

If omitted, CURRENT is assumed.

COLUMN or *C*

Specifies the column to begin the deletion. Values can be an integer or any of the keywords: CURRENT, FIRST_MARK, LAST_MARK, MAXIMUM. Ranges are not allowed.

If omitted, CURRENT is assumed.

Remarks

- For the editor, a word is a string of letters, numbers, or the special characters $, #, @, and _, surrounded by any other characters. The end of a line or beginning of a line is also considered a word boundary.
- If you specify a position that is part of a word, the entire word is deleted.
- If you specify a position that is a blank character, it and all following blanks are deleted.
- If you specify a position that is not part of a word or a blank character, only that character is deleted.
- For more information, see the NOS/VE File Editor manual.

Examples The following deletes the first word in line 50:

```
delete_word line=50 column=1
```

DISPLAY_COLUMN_NUMBERS EDIF Subcommand

Purpose Enables you to list column numbers, which will temporarily overwrite the specified line.

Format

DISPLAY_COLUMN_NUMBERS or
DISCN
ROWS=integer
STATUS=status variable

Parameters *ROWS* or *ROW* or *R*

Specifies which row on the screen is to show the column numbers.

If omitted, the column numbers temporarily overwrite the current line.

Remarks

- The column numbers shown correspond to columns in the file and not column numbers on the screen.
- If the offset is currently nonzero, it is set to zero.
- For more information, see the NOS/VE File Editor manual.

Examples In screen mode, to display the column numbers on the third line, position the cursor on the third line, press Home, and enter:

```
display_column_numbers
```

The following appears:

```
FIRST LINE
SECOND LINE
123456789A123456789B123456789C123456789D123 ...
FOURTH LINE
```

DISPLAY_EDITOR_STATUS EDIF Subcommand

Purpose Enables you to check the status of a number of editor variables including the current tab character, tab columns, and function key definitions.

Format **DISPLAY_EDITOR_STATUS** or **DISES**

STATUS=status variable

Remarks

- The complete command text for each key is not displayed. For the complete command text, see Appendix D in the NOS/VE File Editor manual.
- For more information, see the NOS/VE File Editor manual.

Examples The following example displays the editor status for an editing session in screen mode using a CDC 721 terminal:

```
dises
Press Next/Return for more
 Displaying Editor Status

SCU Editor version is 87314
Modification name is EDIT_FILE
Current file is :NVE.SCU.TEACH
Line width is 0. Search margins are 1 to 256
Set verify option FALSE. State FALSE. No mask character. Tab character is \
Tab columns are:  1 7 72

Function Keys:
  Key       Label       Commands
       F1   Copy        copy_text l=m p=b
 Shift F1   Move        move_text l=m p=b
       F2   Mark        mark_lines
 Shift F2   Unmrk       unmark
       F3   MrkCh       mark_characters
 Shift F3   MrkBx       mark_boxes
       F4   Locate      locate_text t=$si('Enter search string')
 Shift F4   LocNxt      locate_next
       F5   Undo        undo
 Shift F5
       F6   Quit        end
 Shift F6   Exit        esv$text=$si('Reply Y to abandon edit session, or N
       F7   LocAll      esv$text=$si('Enter search string'); if esv$text=''
 Shift F7   Width       if $o>0 then; alis o=0; else; if $noc>80 then; setso
       F8   Break       break_text
 Shift F8   Join        join_text
       F9   SkpEL       position_cursor l=c c=1+$strlen($lt)
 Shift F9
       F10  Middle      align_screen m=c
 Shift F10
```

```
dises
Press Next/Return to complete
 Displaying Editor Status

      F11 Format      format_paragraphs
Shift F11 Center      center_lines
      F12  InsWd      insert_characters nt='
Shift F12  DelWd      delete_word
      F13  InsBk      insert_empty_lines p=b n=$ss-4; position_cursor d=b
Shift F13  DelBk      delete_empty_lines
      F14 Indent      indent_text l=m o=2
Shift F14 Dedent      indent_text l=m o=-2
      F15
Shift F15
      F16
Shift F16
```

DISPLAY_POSITION EDIF Subcommand

Purpose Displays the current line number, current column number, size of the file, and, for screen mode, the line number of the top and bottom line of the screen on the message line.

Format **DISPLAY_POSITION** or
DISP
STATUS=status variable

Remarks For more information, see the NOS/VE File Editor manual.

Examples If, in screen mode, you enter:

```
display_position
```

A display similar to the following appears:

```
Current Line:12  Column:10  Size:109  Top:10  Bottom:18
```

$DISPLAY_UNPRINTABLE_CHARACTERS EDIF Function

Purpose Returns a boolean value indicating whether unprintable ASCII characters displayed at the terminal are replaced by their corresponding mnemonic values (TRUE) or not (FALSE).

Format **$DISPLAY_UNPRINTABLE_CHARACTERS** or
$DUC

Parameters None.

Remarks For further information about functions, see the NOS/VE System Usage manual.

EDIT_FILE
EDIF Subcommand

Purpose Enables you to edit multiple files within the editor.

Format

EDIT_FILE or
EDIF
FILE=**file**
STATUS=*status variable*

Parameters **FILE** or **F**

Specifies the name of the file you want to edit. If the file you specify does not exist, a new file is created.The file must be a sequential file. By default, files created by NOS/VE have this attribute.

The file cannot be an object file.

This parameter is required.

Remarks

- Unlike the EDIT_FILE command, the EDIT_FILE subcommand does not have INPUT, OUTPUT, or PROLOG parameters. Once you are in the editor, you can specify only another file to edit.
- To edit two files on the same screen, use the SET_SCREEN_OPTIONS and EDIT_FILE subcommands.
- For more information, see the NOS/VE File Editor manual.

Examples To edit file BERT, after editing another file, enter:

```
edit_file file=$user.bert
```

END
EDIF Subcommand

Purpose Stops the editor and closes all edited files.

Format

END or
QUIT or
QUI
WRITE_FILE=boolean
STATUS=status variable

Parameters *WRITE_FILE* or *WF*

Specifies if you want changes to all open files made permanent.

If FALSE is specified, no changes are made. If omitted, TRUE is assumed and the changes are permanent.

Remarks For more information, see the NOS/VE File Editor manual.

END_FILE
EDIF Subcommand

Purpose Enables you to close the current file, and continue editing other files.

Format

END_FILE or
ENDF
WRITE_DECK=boolean
STATUS=status variable

Parameters *WRITE_DECK* or *WD* or *WRITE_FILE* or *WF*

Specifies whether to make changes to the current file permanent.

If FALSE is specified, changes are not made permanent.

If omitted, TRUE is assumed.

Remarks

- The END_FILE and END_DECK subcommands can be used interchangeably. (The editor decides on the appropriate action based on the object type (FILE or DECK).
- For more information, see the NOS/VE File Editor manual.

EXCHANGE_POSITION EDIF Subcommand

Purpose Saves the current position in the file you are editing and returns you to a previously saved position.

Format

EXCHANGE_POSITION or
EXCP
STATUS=status variable

Remarks

- You must save a position with the SAVE_POSITION subcommand before executing an EXCHANGE_POSITION subcommand.
- For more information, see the NOS/VE File Editor manual.

EXCHANGE_SCREEN_WIDTH EDIF Subcommand

Purpose Alternates between the 80- and 132-column screen displays, for those terminals that support them.

Format

EXCHANGE_SCREEN_WIDTH or
EXCSW
STATUS=status variable

Remarks

- To set your column width to a value other than 80 or 132, use the SET_SCREEN_OPTIONS subcommand.
- For more information, see the NOS/VE File Editor manual.

Examples If you are using an 80-column screen, entering

```
excsw
```

changes it to a 132-column screen.

FORMAT_PARAGRAPHS
EDIF Subcommand

Purpose Adjusts words or sentences in a paragraph of text to bring line lengths as close as possible to preset margins.

Format

FORMAT_PARAGRAPHS or
FORMAT_PARAGRAPH or
FORP
NUMBER=integer or *keyword*
LINES=range of lines or *keyword*
STATUS=status variable

Parameters *NUMBER* or *N*

Specifies the number of lines to format starting with current line and moving forward. If LINE is omitted and NUMBER is specified, the number of lines in the current paragraph specified by the NUMBER parameter are formatted. If both the NUMBER and LINE parameter are omitted, the current paragraph is assumed.

LINES or *LINE* or *L*

Specifies a range of lines to format. If omitted, the current paragraph is assumed.

Remarks

- Using this subcommand adds two blanks after periods, colons, exclamation marks, and question marks.
- A paragraph consists of any group of lines delimited by empty lines.
- Margins are set using the SET_PARAGRAPH_MARGINS subcommand.
- If you have not entered the SET_PARAGRAPH_MARGINS subcommand, the paragraph margins are set at 1 and 65. The first line is indented four characters.
- For more information, see the NOS/VE File Editor manual.

Examples The following example adjusts the current line and the 5 subsequent lines to conform to previously set margins.

```
format_paragraph number=6
```

$FUNCTION_ROW
EDIF Function

Purpose Returns an integer specifying the top row in which the menu of operations is displayed.

Format **$FUNCTION_ROW** or
$FR

Parameters None.

Remarks

- If you are in line mode, zero is returned.
- For more information, see the NOS/VE File Editor manual.

$FUNCTION_SIZE
EDIF Function

Purpose Returns an integer specifying the number of rows on the screen used by the menu of operations.

Format **$FUNCTION_SIZE** or
$FS

Parameters None.

Remarks

- If you are in line mode, zero is returned.
- For more information, see the NOS/VE File Editor manual.

Examples

- The following commands display the number of screen rows required to display a single menu row.

```
/set_screen_options mr=1
/display_value $function_size
2
```

- The following command, executed repeatedly, will display 0, 1, and 2 rows of the menu of operations.

```
setso mr=$mod($function_size/2+1,3)
```

$HOME_ROW
EDIF Function

Purpose Returns an integer specifying the row used for entering subcommands and responses to the editor.

Format
$HOME_ROW or
$HR

Parameters None.

Remarks
- If you are in line mode, zero is returned.
- For more information, see the NOS/VE File Editor manual.

INDENT_TEXT
EDIF Subcommand

Purpose Inserts blank characters or deletes characters in front of lines of text.

Format
INDENT_TEXT or
INDT
OFFSET=integer
NUMBER=integer or *keyword*
LINES=range of lines or *keyword*
STATUS=status variable

Parameters *OFFSET* or *O*

Specifies the number of columns to indent the specified block of text.

If positive, that number of blanks are added.

If negative, that number of characters are deleted.

If omitted, 1 is assumed.

NUMBER or *N*

Specifies the number of lines to be indented.

If you specify a range for the LINES parameter, the NUMBER parameter assumes a value of ALL.

If no range is specified for the LINES parameter, the NUMBER parameter assumes a value of 1.

LINES or *LINE* or *L*

Specifies a range of lines to be indented. Values can be an integer, line identifier, or one of the keywords: ALL, CURRENT, FIRST, FIRST_MARK, FIRST_SCREEN, LAST, LAST_MARK, LAST_SCREEN, MARK, SCREEN.

If no range is specified, range is the current line to the last line.

Remarks For more information, see the NOS/VE File Editor manual.

Examples

- The following example indents all lines five spaces:

```
indt offset=5 line=all
```

- The following example deletes the first 7 characters from lines 25 through the last line:

```
indent_text offset=-7 line=25..last
```

INSERT_CHARACTERS EDIF Subcommand

Purpose Inserts a string of characters before a specified location.

Format

INSERT_CHARACTERS or
INSC or
INSERT_CHARACTER or
IC
NEW_TEXT=string
INSERTION_LOCATION=lines or *keyword*
INSERTION_COLUMN=integer or *keyword*
STATUS=status variable

Parameters *NEW_TEXT* or *NT*

Specifies the text to be inserted. The text must be enclosed in apostrophes.

If you omit this parameter, one blank character is inserted.

INSERTION_LOCATION or *IL*

Specifies the line in which the text is to be inserted. Values can be an integer, line identifier, or one of the keywords: CURRENT, FIRST, FIRST_MARK, FIRST_SCREEN, LAST, LAST_MARK, LAST_SCREEN. Ranges are not allowed.

If omitted, the value is the current line.

INSERTION_COLUMN or *IC*

Specifies the column before which the insertion is to occur. Values can be an integer from 1 through 256 or any of the keywords: CURRENT, FIRST_MARK, LAST_MARK, MAXIMUM. Ranges are not allowed.

If you omit this parameter, the current column is assumed.

Remarks For more information, see the NOS/VE File Editor manual.

Examples

- The following inserts the text *Short Comment* in front of the current column on the current line:

```
insert_characters 'Short Comment'
```

- Using an SCL string variable as the value for the NEW_TEXT parameter is an efficient way of inserting the same text numerous places in a file. For example, you could initialize a string variable as follows:

```
a = 'characters to be inserted'
```

When the cursor is positioned at a point where you want to insert the text, simply enter the following command:

```
insert_characters a
```

INSERT_EMPTY_LINES EDIF Subcommand

Purpose Enables you to insert empty lines.

Format

INSERT_EMPTY_LINES or
INSERT_EMPTY_LINE or
INSEL
NUMBER=integer
INSERTION_LOCATION=lines or *keyword*
PLACEMENT=keyword
STATUS=status variable

Parameters *NUMBER* or *N*

Specifies the number of empty lines to be inserted.

If omitted, 1 is assumed.

INSERTION_LOCATION or *IL*

Specifies the line before or after which the insertion is to occur. Values can be an integer, line identifier, or one of the keywords: CURRENT, FIRST, FIRST_MARK, FIRST_SCREEN, LAST, LAST_MARK, LAST_SCREEN. Ranges are not allowed.

If omitted, CURRENT is assumed.

PLACEMENT or *P*

Specifies whether the insertion is to occur BEFORE(B) or AFTER(A) the line specified by the INSERTION_LOCATION parameter.

If omitted, AFTER is assumed.

Remarks For more information, see the NOS/VE File Editor manual.

Examples The following inserts 2 empty lines after line 50:

```
insel number=2 insertion_location=50
```

INSERT_LINES
EDIF Subcommand

Purpose Inserts one or more lines of text.

Format

INSERT_LINES or
INSERT_LINE or
INSL or
I

NEW_TEXT=string
PLACEMENT=keyword
INSERTION_LOCATION=insertion_location or
keyword
UNTIL=string
STATUS=status variable

Parameters

NEW_TEXT or *NT*

Specifies the new line of text to be inserted.

If NEW_TEXT is omitted, the text to be inserted is taken from the command input file.

PLACEMENT or *P*

Indicates whether the insertion is to occur BEFORE (B) or AFTER (A) the location specified by the INSERTION_LOCATION parameter.

If omitted, AFTER is assumed.

INSERTION_LOCATION or *IL*

Specifies the line after which or before which the insertion is to occur. Values can be an integer, line identifier, or one of the keywords: CURRENT, FIRST, FIRST_MARK, FIRST_SCREEN, LAST, LAST_MARK, LAST_SCREEN. Ranges are not allowed.

If omitted, CURRENT is assumed.

UNTIL or *U*

In line mode, specifies a string that stops the insert.

If the NEW_TEXT parameter is omitted, you are prompted to enter input until the editor encounters the string specified by this parameter at the end of a line.

If the UNTIL parameter is omitted, ** is assumed.

Remarks

For more information, see the NOS/VE File Editor manual.

Examples

- The following inserts the line *NEW LINE* after the current line.

```
insert_lines 'NEW LINE'
```

- The following inserts the line *Insert* before the current line:

```
insert_lines new_text='Insert' placement=before
```

- The following inserts the line *First line* before the first line of the file.

```
insl nt='First line' insertion_location=first p=b
```

- The following inserts lines from the command input file before line 45 until a # character is encountered as the last character in a line.

```
i il=45 p=b until='#'
```

INSERT_WORD
EDIF Subcommand

Purpose Inserts a string or 32 blank characters.

Format

INSERT_WORD or
INSW or
IW
NEW_TEXT=string
INSERTION_LOCATION=lines or *keyword*
INSERTION_COLUMN=integer or *keyword*
STATUS=status variable

Parameters

NEW_TEXT or *NT*

Specifies the string to be inserted. The default is 32 blanks.

INSERTION_LOCATION or *IL*

Specifies the line in which the string is to be inserted. Values can be an integer, line identifier, or one of the keywords: CURRENT, FIRST, FIRST_MARK, FIRST_SCREEN, LAST, LAST_MARK, LAST_SCREEN. Ranges are not allowed.

If omitted, value is the current line.

INSERTION_COLUMN or *IC*

Specifies the column before which the insertion is to occur. Values can be an integer from 1 through 256 or any of the keywords: CURRENT, FIRST_MARK, LAST_MARK, MAXIMUM.

If omitted, CURRENT is assumed.

Remarks For more information, see the NOS/VE File Editor manual.

Examples

- The following inserts the word *LINE* in front of line 10:

```
insw new_text='LINE' insertion_location=10 ..
   insertion_column=1
```

- The following inserts 32 blank characters before the current column of the current line:

```
insert_word
```

JOIN_TEXT EDIF Subcommand

Purpose Joins a line with the next line by appending the second to the first.

Format

JOIN_TEXT or
JOIT or
J

LINES=lines or *keyword*
COLUMN=integer or *keyword*
STATUS=status variable

Parameters

LINES or *LINE* or *L*

Specifies the first of two lines to be joined. The next line is joined to the specified line. Values can be an integer, line identifier, or one of the keywords: CURRENT, FIRST, FIRST_MARK, FIRST_SCREEN, LAST, LAST_MARK, LAST_SCREEN. Ranges are not allowed.

If omitted, CURRENT is assumed.

COLUMN or *C*

Specifies the starting column to which the second line is moved. The second line is always added after the end of the first line. The columns parameter determines how far after the first line the second line is added.

Values can be an integer from 1 through 256 or any of the keywords: CURRENT, FIRST_MARK, LAST_MARK, MAXIMUM. Ranges are not allowed.

If the value you specify is less than or equal to the length of the first line, the line is added to the end of the first line. If the value you specify is greater than the length of the first line, the editor fills the columns in between with blank characters.

If COLUMN is omitted, CURRENT is assumed.

Remarks

- If the joined line is longer than 256 characters, the subcommand is not performed and the editor displays the following message:

  ```
  Line length exceeded.
  ```

- For more information, see the NOS/VE File Editor manual.

$LINE_IDENTIFIER EDIF Function

Purpose Returns a string specifying the line identifier of the current line (for editing decks only).

Format **$LINE_IDENTIFIER** or
$LI

Parameters None.

Remarks For more information, see the NOS/VE File Editor manual.

$LINE_TEXT EDIF Function

Purpose Returns the text of the current line as a string.

Format **$LINE_TEXT** or
$LT

Parameters None.

Remarks

- One of the uses for this function is in procedures that operate on lines of a file. You can use the POSITION_CURSOR subcommand to move to a line, and then use the $LINE_TEXT function to return the value of the line.

- For more information, see the NOS/VE File Editor manual.

Examples

- The following adds the string 'append' to the end of the current line:

```
replace_line nt=$lt//'append'
```

- The following expression returns the length of the current line:

```
$strlen($line_text)
```

- The following executes the current line:

```
include_line sl=$lt
```

LIST_BACKWARDS EDIF Subcommand

Purpose

In line mode, displays a range of lines ending with the current line. In effect, it enables you to view a number of lines just before the current line and end up where you started.

Format

LIST_BACKWARDS or
LISB or
LIST_BACKWARD or
LB
NUMBER=integer or *keyword*
STATUS=status variable

Parameters

NUMBER or *N*

Specifies the number of lines to list. Values can be integers or the keyword ALL. ALL lists all lines from the beginning of the file to the current line.

If NUMBER is omitted, a value of 1 is assumed.

Remarks

- This subcommand is typically only used in line mode.
- For more information, see the NOS/VE File Editor manual.

Examples

The following example lists 15 lines ending with the current line.

```
list_backward n=15
```

LIST_FORWARDS
EDIF Subcommand

Purpose In line mode, displays a range of lines beginning with the current line.

Format

LIST_FORWARDS or
LISF or
LIST_FORWARD or
LF
NUMBER=integer or *keyword*
STATUS=status variable

Parameters *NUMBER* or *N*

Specifies the number of lines to list. Values can be integers or the keyword ALL. ALL lists all lines from the current line to the end of the file.

If NUMBER is omitted, a value of 1 is assumed.

Remarks

- This subcommand is typically used only in line mode.
- For more information, see the NOS/VE File Editor manual.

Examples The following example lists 15 lines beginning with the current line.

```
list_forward n=15
```

LIST_LINES
EDIF Subcommand

Purpose In line mode, lists a specified line or range of lines. In screen mode, the cursor is positioned at the specified line, or the last line in the range.

Format

LIST_LINES or
LISL or
LIST_LINE or
LL
LINES=range of lines or *keyword*
STATUS=status variable

Parameters *LINES* or *LINE* or *L*

Specifies the line or range of lines to list. Values can be an integer, line identifier, or one of the keywords: ALL, CURRENT, FIRST, FIRST_MARK, FIRST_SCREEN, LAST, LAST_MARK, LAST_SCREEN, MARK, SCREEN.

If LINE is omitted, CURRENT is assumed.

Remarks For more information, see the NOS/VE File Editor manual.

Examples The following example lists lines 25 through 40.

```
list_lines l=25..40
```

LOCATE_ALL
EDIF Subcommand

Purpose Searches the entire file to locate all occurrences of a specified string. In screen mode, all occurrences are then listed in a directory enabling you to either position the cursor at a specific line or enter the desired line number. In line mode, all occurrences are listed and you are positioned at the last occurrence of the string.

Format **LOCATE_ALL** or
LOCA or
LA
TEXT=range of string
STATUS=status variable

Parameters *TEXT* or *T*

Specifies the text string you want to find. If omitted, the last text string specified is assumed, if any.

Remarks For more information, see the NOS/VE File Editor manual.

Examples The following example locates all occurrences of the string, *find this text*, in the file and lists them.

```
locate_all text='find this text'
```

LOCATE_EMPTY_LINES EDIF Subcommand

Purpose Finds empty lines. An empty line is a line of all blank characters.

Format

LOCATE_EMPTY_LINES or
LOCATE_EMPTY_LINE or
LOCEL
NUMBER=integer or *keyword*
LINES=range of lines or *keyword*
DIRECTION=keyword
VETO=boolean
STATUS=status variable

Parameters

NUMBER or *N*

Specifies the number of empty lines to find. Values can be numbers or the keyword ALL.

If a LINE parameter is specified, NUMBER assumes a value of ALL.

If no LINE parameter is specified, NUMBER assumes a value of 1.

LINES or *LINE* or *L*

Specifies a range of lines to search.

Values can be an integer, line identifier, or one of the keywords: ALL, CURRENT, FIRST, FIRST_MARK, FIRST_SCREEN, LAST, LAST_MARK, LAST_SCREEN, MARK, SCREEN. If you specify a value of only one line, the search is limited to that line.

If you omit LINE and specify BACKWARD for the DIRECTION parameter, CURRENT..FIRST is assumed.

If both LINE and DIRECTION are omitted, CURRENT..LAST is assumed.

DIRECTION or *D*

Specifies whether to search FORWARD (F) or BACKWARD (B) from the current line.

If no value is specified, FORWARD is assumed.

VETO or *V*

Instructs the editor to turn the veto option on or off.

If TRUE is specified, the editor displays a directory of located lines.

If VETO is omitted, FALSE is assumed.

Remarks For more information, see the NOS/VE File Editor manual.

Examples

- The following positions the cursor to the fifth empty line:

```
locate_empty_lines number=5
```

- The following positions the cursor to the last empty line:

```
locel line=20..40
```

- The following positions the cursor to the tenth empty line in the marked text:

```
locel number=10 line=mark
```

LOCATE_NEXT EDIF Subcommand

Purpose Locates the next occurrence of a previously specified string. The search begins one column after the current column.

Format

LOCATE_NEXT or
LOCN or
LN
STATUS=status variable

Remarks For more information, see the NOS/VE File Editor manual.

LOCATE_STRING EDIF Subcommand

Purpose Beginning at the current line and column, it searches for the specified string.

Format

LOCATE_STRING or
LOCS or
LS
TEXT=range of string
STATUS=status variable

Parameters *TEXT* or *T*

Specifies strings of text in the first and last lines of a block of text to be located. If you enter only one string, the block of text to be located will contain only one line. If you enter two strings, the search for the second begins immediately after the first is found and the cursor is positioned at the beginning of the first string.

If omitted, the last string parameter specified, if any, is used.

Remarks

- This subcommand is typically only used in line mode.
- For more information, see the NOS/VE File Editor manual.

Examples The following example locates the string *work now*:

```
locate_string 'work now'
```

LOCATE_TEXT EDIF Subcommand

Purpose Locates blocks of text.

Format

LOCATE_TEXT or
LOCT or
L
TEXT=range of string
NUMBER=integer or *keyword*
LINES=range of lines or *keyword*
COLUMNS=range of integer or *keyword*
BOUNDARY=keyword
DIRECTION=keyword

UPPER_CASE=boolean
WORD=boolean
REPEAT_SEARCH=boolean
VETO=boolean
STATUS=status variable

Parameters

TEXT or *T*

Specifies strings of text in the first and last lines of a block of text to be located. If you enter only one string, the block of text to be located will contain only one line. If you enter two strings, the search for the second begins immediately after the first is found and the cursor is positioned at the beginning of the first string.

If TEXT is omitted, the lines to be located will be determined by the NUMBER, LINE, and DIRECTION parameters.

NUMBER or *N*

Specifies the number of blocks of text to be found. Values for this parameter can be an integer or the keyword ALL(A).

In line mode, use the NUMBER parameter to display a range of lines.

If you specify a range of values for the LINE parameter, NUMBER assumes a value of ALL.

If no range is specified for the LINE parameter, NUMBER assumes a value of 1.

LINES or *LINE* or *L*

Specifies a range of lines to be searched.

Values can be an integer, line identifier, or any of the keywords: ALL, CURRENT, FIRST, FIRST_MARK, FIRST_SCREEN, LAST, LAST_MARK, LAST_SCREEN, MARK, SCREEN. If one line is specified, the search is limited to that line.

In line mode, use the LINE parameter to specify which lines to print.

If you omit LINE and specify BACKWARD for the DIRECTION parameter, CURRENT..FIRST is assumed.

If LINE and DIRECTION are both omitted, CURRENT..LAST is assumed.

COLUMNS or *COLUMN* or *C*

Specifies the range of columns to search. Values can be an integer from 1 through 256 or any of the keywords: CURRENT, FIRST_MARK, LAST_MARK, MARK, MAXIMUM.

If omitted, CURRENT is assumed. If omitted and you have specified LINE=MARK, the marked lines provide the column boundaries. If COLUMN is omitted and you have specified a LINE parameter other than MARK, all columns will be searched.

BOUNDARY or *B*

Specifies the type of boundary that will limit the search. Values can be LINE or STREAM.

If COLUMNS is specified and BOUNDARY is omitted, STREAM is assumed.

If both BOUNDARY and COLUMNS are omitted, LINE is assumed.

DIRECTION or *D*

Specifies whether to search FORWARD (F) or BACKWARD (B) from the current line.

If no value is specified, FORWARD is assumed.

UPPER_CASE or *UC*

Determines the significance of capitalization in the search.

If you specify TRUE, the editor searches the file as if it were all uppercase.

If you specify FALSE, the editor searches for the text exactly as it was entered.

If omitted, FALSE is assumed unless you specify TRUE for REPEAT_SEARCH. In this case, your last value for UPPER_CASE is used.

WORD or *W*

Determines whether the editor searches for the specified text string as a word (the text you want to search for is surrounded by nonalphanumeric characters).

If you specify TRUE, the editor searches for the text as a word. If you specify FALSE, it doesn't.

If omitted, FALSE is assumed unless you specify TRUE for REPEAT_SEARCH. In this case, your last value for WORD is used.

REPEAT_SEARCH or *RS*

Instructs the editor how to use the values entered for the last TEXT, UPPER_CASE, and WORD parameters.

If you specify TRUE, the editor uses the same TEXT, UPPER_CASE, and WORD parameters as the last time you entered them for any subcommand (unless you have specified values for this subcommand).

If you specify FALSE, the editor uses the parameters entered with the current subcommand.

If omitted, FALSE is assumed.

VETO or *V*

Instructs the editor to turn the veto option on or off.

If TRUE is specified, the editor displays a directory of located lines.

If VETO is omitted, FALSE is assumed.

Remarks For more information, see the NOS/VE File Editor manual.

Examples

- The following example locates the next occurrence of *PROCEND*:

```
locate_text 'PROCEND'
```

- The following example locates the previous occurrence of *TITLE*:

```
locate_text 'TITLE' direction=backward
```

- The following example positions the cursor on line 250 of the current file or deck:

```
loct line=250
```

- The following example locates the string you last specified as a value for the TEXT parameter:

```
loct repeat_search=true
```

- The following example locates all occurrences of PARAMETER from the current position to the end of the file and displays the lines in a directory-type display:

```
l 'PARAMETER' number=all veto=true
```

- The following example locates the next block of text beginning with *one* and ending with *twenty*:

```
l 'one'..'twenty'
```

- The following example prints the current line and four subsequent lines in line mode. In screen mode, the cursor is positioned four lines forward:

```
l n=5
```

- The following example displays a directory of each occurrence of the word *MISPELL* regardless of case:

```
l 'mispell' l=a uc=true v=true
```

LOCATE_WIDE_LINES EDIF Subcommand

Purpose Locates lines that are wider than the margins set by the SET_LINE_WIDTH subcommand.

Format

LOCATE_WIDE_LINES or
LOCATE_WIDE_LINE or
LOCWL
NUMBER=integer or *keyword*
LINES=range of lines or *keyword*
DIRECTION=keyword
VETO=boolean
STATUS=status variable

Parameters *NUMBER* or *N*

Specifies the number of wide lines to be found. Values for this parameter can be an integer or the keyword ALL(A).

If a LINES parameter is specified, NUMBER assumes a value of ALL. Otherwise, the assumed value for NUMBER is 1.

LINES or *LINE* or *L*

Specifies a range of lines to be searched. Values can be an integer, line identifier, or one of the keywords: ALL, CURRENT, FIRST, FIRST_MARK, FIRST_SCREEN, LAST, LAST_MARK, LAST_SCREEN, MARK, SCREEN. If you specify a value for only one line, the search is limited to that line.

If LINE and DIRECTION are omitted, CURRENT..LAST is assumed. If you omit LINE and specify BACKWARD for the DIRECTION parameter, CURRENT..FIRST is assumed.

DIRECTION or *D*

Specifies whether to search FORWARD (F) or BACKWARD (B) from the current line.

If omitted, FORWARD is assumed.

VETO or *V*

Instructs the editor to turn the veto option on or off.

If TRUE is specified, the editor displays a directory of located lines.

If VETO is omitted, FALSE is assumed.

Remarks

For more information, see the NOS/VE File Editor manual.

Examples

- The following locates the first wide line in the file:

```
locate_wide_lines number=1 lines=all
```

- The following locates the next wide line starting at the current line:

```
locwl
```

- The following locates and displays a directory of all wide lines between the top line of the current screen and the last line of the file:

```
locwl line=first_screen..last veto=true
```

MARK_BOXES
EDIF Subcommand

Purpose Marks a rectangular area of text.

Format

MARK_BOXES or
MARB or
MARK_BOX or
MB
LINES=range of lines or *keyword*
COLUMNS=range of integer or *keyword*
STATUS=status variable

Parameters *LINES* or *LINE* or *L*

Specifies the lines in which the corners of the box reside. Values can be an integer, line identifier, or one of the keywords: ALL, CURRENT, FIRST, FIRST_MARK, FIRST_SCREEN, LAST, LAST_MARK, LAST_SCREEN, MARK, SCREEN.

If omitted, CURRENT is assumed.

COLUMNS or *COLUMN* or *C*

Specifies the columns in which the corners of the box reside. Values can be any integer from 1 through 256 or any of the keywords: CURRENT, FIRST_MARK, LAST_MARK, MARK, MAXIMUM.

If omitted, CURRENT is assumed.

Remarks

- Currently, the only operations supported for MARK_BOX are the following functions:

  $MARK_FIRST_COLUMN
  $MARK_FIRST_LINE
  $MARK_LAST_COLUMN
  $MARK_LAST_LINE
  $MARK_TYPE (returns a value of BOX)

  These allow you to implement your own SCL procedures to operate on the rectangular area of text. None of the CDC-supplied editor subcommands support box marks.

- For more information, see the NOS/VE File Editor manual.

Examples

- An example procedure called MOVE_BOX that moves a marked box can be found in the online Examples manual.

- The following example marks a box 5 lines by 1 column; the marked area will cover lines 4 through 8 at column 12:

```
mark_box lines=4..8 column=12
```

- To mark a box with dimensions 5 lines by 3 columns, enter:

```
mark_box lines=2..6 columns=3..5
```

The marked area covers lines 2, 3, 4, 5, and 6 at columns 3, 4, and 5 in each line as illustrated below:

```
    1 2 3 4 5 6 7 8

1   x x x x x x x x
2   x x x x x x x x
3   x x x x x x x x
4   x x x x x x x x
5   x x x x x x x x
6   x x x x x x x x
7   x x x x x x x x
```

The same results can be achieved by positioning the cursor to the upper left corner of the intended box (line 2, column 3), entering the MARK_BOX subcommand, then positioning the cursor to the lower right corner of the intended box, (line 6, column 5) and entering the MARK_BOX subcommand.

- The following procedure deletes a marked box:

```
PROC delete_box, db (
   status : var of status = $optional)
   IF $mark_type <> 'BOX' THEN
      put_row 'No box has been marked.' r=$mr
      EXIT_PROC
   IFEND
   FOR box_line=$mark_first_line ..
      TO $mark_last_line DO
      delete_text line=box_line ..
         column=$mark_first_column..$mark_last_column
   FOREND
   unmark
PROCEND delete_box
```

MARK_CHARACTERS EDIF Subcommand

Purpose Marks specific characters. These marks specify the boundary for text that is to be processed later by subcommands that insert, delete, move, copy, and replace text.

Format

MARK_CHARACTERS or
MARC or
MARK_CHARACTER or
MC
LINES=range of lines or *keyword*
COLUMNS=range of integer or *keyword*
STATUS=status variable

Parameters *LINES* or *LINE* or *L*

Specifies the lines in which the marked characters reside.

If LINES is omitted, CURRENT is assumed.

COLUMNS or *COLUMN* or *C*

Specifies the columns to be marked within the specified line(s). Values can be any integer from 1 through 256 or any of the keywords: CURRENT, FIRST_MARK, LAST_MARK, MARK, MAXIMUM.

If COLUMN is omitted, CURRENT is assumed.

Remarks

- If a character is specified, only that character is marked. If a single character is specified and another single character is already marked, the characters between the two will become marked. If a range is specified, the entire range is marked and any other marks are unmarked.

- Even though you can mark a range of characters by entering two MARK_CHARACTER subcommands, if you mark one character and then reference the mark in another editing operation such as inserting or copying, the editor assumes the marking operation is complete. Then, when you mark a second character, the editor starts another marking operation; the first character is unmarked, and the second marked.

- The following functions can be used to determine the location and type of the marked region.

  $MARK_FIRST_COLUMN
  $MARK_FIRST_LINE
  $MARK_LAST_COLUMN
  $MARK_LAST_LINE
  $MARK_TYPE (returns a value of STREAM)

- For more information, see the NOS/VE File Editor manual.

Examples

- The following marks column 30 of line 40 through column 30 of line 50:

```
mark_character line=40..50 column=30
```

- The following marks columns 7 through 10 of the current line:

```
mark_character column=7..10
```

- To mark column 5 of line 2 through column 3 of line 5, enter:

```
mark_character  lines=2..5 columns=5..3
```

  The marked area covers column 5 of line 2 through column 3 of line 5 as illustrated below:

```
    1 2 3 4 5 6 7 8 ...

1   x x x x x x x x
2   x x x x x x x x
```

```
3   x x x x x x x x
4   x x x x x x x x
5   x x x x x x x x
6   x x x x x x x x
7   x x x x x x x x
```

The same results can be achieved by positioning the cursor to the first character to be marked (line 2, column 5), entering the MARK_CHARACTER subcommand, then positioning the cursor to the last character to be marked (line 5, column 3) and entering the MARK_CHARACTER subcommand.

$MARK_FIRST_COLUMN EDIF Function

Purpose Returns an integer specifying the column number of the first marked column.

Format **$MARK_FIRST_COLUMN** or **$MFC**

Parameters None.

Remarks For more information, see the NOS/VE File Editor manual.

$MARK_FIRST_LINE EDIF Function

Purpose Returns an integer specifying the line number of the first marked line.

Format **$MARK_FIRST_LINE** or **$MFL**

Parameters None.

Remarks For more information, see the NOS/VE File Editor manual.

$MARK_LAST_COLUMN
EDIF Function

Purpose Returns an integer specifying the column number of the last marked column.

Format **$MARK_LAST_COLUMN** or
$MLC

Parameters None.

Remarks For more information, see the NOS/VE File Editor manual.

$MARK_LAST_LINE
EDIF Function

Purpose Returns an integer specifying the line number of the last marked line.

Format **$MARK_LAST_LINE** or
$MLL

Parameters None.

Remarks For more information, see the NOS/VE File Editor manual.

Examples The following statements write marked lines to file $LOCAL.SCR1. If no lines (or one line) are marked, all lines are written to the file.

```
lines_to_write = 'all'
IF $mark_first_line<>$mark_last_line THEN
   lines_to_write = 'mark'
IFEND
write_file f=$local.scr1 l=lines_to_write
```

MARK_LINES
EDIF Subcommand

Purpose Marks a line to be processed later.

Format

MARK_LINES or
MARK_LINE or
MARL or
ML
LINES=range of lines or *keyword*
STATUS=status variable

Parameters *LINES* or *LINE* or *L*

Specifies a line or range of lines to be marked. Marked text can be processed by subcommands that insert, delete, move, copy, and replace text. Values can be an integer, line identifier, or one of the keywords: ALL, CURRENT, FIRST, FIRST_MARK, FIRST_SCREEN, LAST, LAST_MARK, LAST_SCREEN, MARK, SCREEN.

If LINE is omitted, the current line is assumed.

Remarks

- If a line is specified, only that line is marked. If a single line is specified and another single line is already marked, the lines between the two will become marked. If a range is specified, the entire range is marked and any other marks are unmarked.

- Even though you can mark a range of lines by entering two MARK_LINES subcommands, if you mark one line and then reference the mark in another editing operation such as inserting or copying, the editor assumes the marking operation is complete. Then, when you mark a second line, the editor starts another marking operation; the first line is unmarked, and the second marked.

- The following functions can be used to determine the location and type of the marked region.

  $MARK_FIRST_LINE
  $MARK_LAST_LINE
  $MARK_TYPE (returns a value of LINES)

- For more information, see the NOS/VE File Editor manual.

$MARK_OBJECT
EDIF Function

Purpose Returns a string specifying the name of the current file or deck containing the marked text.

Format **$MARK_OBJECT** or
$MO

Parameters None.

Remarks

- Use the $MARK_OBJECT_TYPE function to determine if the object is a file or a deck.
- For more information, see the NOS/VE File Editor manual.

$MARK_OBJECT_TYPE
EDIF Function

Purpose Returns a string specifying if the marked text is in a file or a deck. Values returned can be FILE, DECK, or NULL.

Format **$MARK_OBJECT_TYPE** or
$MOT

Parameters None.

Remarks For more information, see the NOS/VE File Editor manual.

$MARK_TYPE
EDIF Function

Purpose Returns a value indicating whether the marked region is bounded by lines, bounded by characters, or is a box. Values returned can be LINES (line boundary), STREAM (character boundary), or BOX.

Format **$MARK_TYPE** or
$MT

Parameters None.

Remarks For more information, see the NOS/VE File Editor manual.

$MESSAGE_ROW
EDIF Function

Purpose Returns an integer specifying the number of the row on the screen used to display messages.

Format

$MESSAGE_ROW or
$MR

Parameters None.

Remarks

- If you are in line mode, zero is returned.
- This function is often used as the value for the ROW parameter on the PUT_ROW subcommand.
- For more information, see the NOS/VE File Editor manual.

MOVE_TEXT
EDIF Subcommand

Purpose Moves a block of text from one place to another in the same file.

Format

MOVE_TEXT or
MOVT or
M

TEXT=range of string
NUMBER=integer or *keyword*
LINES=range of lines or *keyword*
COLUMNS=range of integer or *keyword*
INSERTION_LOCATION=lines or *keyword*
INSERTION_COLUMN=integer or *keyword*
PLACEMENT=keyword
BOUNDARY=keyword
UPPER_CASE=boolean
WORD=boolean
REPEAT_SEARCH=boolean
STATUS=status variable

Parameters

TEXT or *T*

Specifies string(s) of text in the first and last lines of a block of text to be moved. If you enter only one string, the block of text to be moved will contain only one line. If you enter two strings, the search for the second begins immediately after the first is found.

If TEXT is specified, the INSERTION_COLUMNS and BOUNDARY parameters are ignored and line boundaries are used.

If omitted, the lines to be copied will be determined by the NUMBER and LINES parameters or the REPEAT_SEARCH parameter.

NUMBER or *N*

Specifies the number of blocks of text to be moved. Values for this parameter can be numbers or the keyword ALL (A).

If omitted and a range is specified for the LINES parameter, this parameter assumes a value of ALL. Otherwise, the assumed value is 1.

LINES or *LINE* or *L*

Specifies the range of lines to be searched for the text to be moved. If a single value is specified, only that line is searched. Values can be an integer, line identifier, or one of the keywords: ALL, CURRENT, FIRST, FIRST_MARK, FIRST_SCREEN, LAST, LAST_MARK, LAST_SCREEN, MARK, SCREEN.

If omitted, CURRENT..LAST is assumed.

COLUMNS or *COLUMN* or *C*

Specifies the range of columns to be searched for text to be moved. The integers can be from 1 through 256 or any of the keywords: CURRENT, FIRST_MARK, LAST_MARK, MARK, MAXIMUM.

If omitted, CURRENT is assumed. If omitted and you have specified LINE=MARK, the marked lines provide the column boundaries. If COLUMN is omitted and you have specified a LINE parameter other than MARK, all columns will be searched.

INSERTION_LOCATION or *IL*

Specifies the line before or after which the text is to be moved (depending on the value of the PLACEMENT parameter). Values can be an integer, line identifier, or one of the LINE keywords: CURRENT, FIRST, FIRST_MARK, FIRST_SCREEN, LAST, LAST_MARK, LAST_SCREEN. Ranges are not allowed.

If omitted, CURRENT is assumed.

INSERTION_COLUMN or *IC*

Specifies the column before or after which the text is to be moved (depending on the value of the PLACEMENT parameter). Values can be an integer from 1 through 256, or any of the COLUMN keywords: CURRENT, FIRST_MARK, LAST_MARK, MAXIMUM. Ranges are not allowed.

If omitted, CURRENT is assumed.

If a value for TEXT is specified, INSERTION_COLUMN is ignored.

PLACEMENT or *P*

Specifies if the moved lines are to appear BEFORE (B) or AFTER (A) the location specified by the INSERTION_LOCATION parameter.

If omitted, AFTER is assumed.

BOUNDARY or *B*

Specifies the type of boundary that will limit the search. Values can be LINE or STREAM.

If COLUMNS is specified and BOUNDARY is omitted, STREAM is assumed.

If both BOUNDARY and COLUMNS are omitted, LINE is assumed.

If a value for TEXT is specified, BOUNDARY is ignored; line boundaries are used.

UPPER_CASE or *UC*

Determines the significance of capitalization in the search.

If you specify TRUE, the editor searches the file as if it were all uppercase.

If you specify FALSE, the editor searches for the text exactly as it was entered.

If omitted, FALSE is assumed unless you specify TRUE for REPEAT_SEARCH. In this case, your last value for UPPER_CASE is used.

WORD or *W*

Determines whether the editor searches for the specified text string as a word (the text you want to search for is surrounded by nonalphanumeric characters).

If you specify TRUE, the editor searches for the text as a word. If you specify FALSE, it doesn't.

If omitted, FALSE is assumed unless you specify TRUE for REPEAT_SEARCH. In this case, your last value for WORD is used.

REPEAT_SEARCH or *RS*

Instructs the editor how to use the values entered for the last TEXT, UPPER_CASE, and WORD parameters.

If you specify TRUE, the editor uses the same TEXT, UPPER_CASE, and WORD parameters as the last time you entered them for any subcommand (unless you have specified values for this subcommand).

If you specify FALSE, the editor uses the parameters entered with the current subcommand.

If omitted, FALSE is assumed.

Remarks For more information, see the NOS/VE File Editor manual.

Examples

- The following moves lines 30 through 40 to immediately after the current line.

```
move_text line=30..40
```

- The following moves the next occurrence of a block of text beginning with the line containing *orchid* and ending with the line containing *violet* to immediately before line 71:

```
movt text='orchid'..'violet' il=71 placement=before
```

$NEW_TEXT
EDIF Function

Purpose Returns the last string entered in a NEW_TEXT parameter.

Format **$NEW_TEXT** or
$NT

Parameters None.

Remarks For more information, see the NOS/VE File Editor manual.

Examples If you enter:

```
replace_text text='good' new_text='the best'
```

you can then use:

```
r t='better' nt=$nt
```

to replace *better* with *the best.*

$NUMBER_OF_COLUMNS
EDIF Function

Purpose Returns an integer specifying the number of columns currently being used to display text on the screen.

Format **$NUMBER_OF_COLUMNS** or
$NUMBER_OF_COLUMN or
$NOC

Parameters None.

Remarks

- If you are in line mode, zero is returned.
- For more information, see the NOS/VE File Editor manual.

$NUMBER_OF_ROWS
EDIF Function

Purpose Returns an integer specifying the number of rows currently displayed on the screen, including the menu of operations, message line, home line, and file header.

Format **$NUMBER_OF_ROWS** or
$NUMBER_OF_ROW or
$NOR

Parameters None.

Remarks

- If you are in line mode, zero is returned.
- For more information, see the NOS/VE File Editor manual.

$NUMBER_OF_SPLITS
EDIF Function

Purpose Returns an integer specifying the number of splits on the screen.

Format **$NUMBER_OF_SPLITS** or
$NUMBER_OF_SPLIT or
$NOS

Parameters None.

Remarks

- If you are in line mode, zero is returned.
- For more information, see the NOS/VE File Editor manual.

Examples The following alternates between 1 or 2 screens:

```
set_screen_option s=3-$number_of_splits
```

$OFFSET
EDIF Function

Purpose Returns an integer identifying the number specified on the OFFSET parameter of the ALIGN_SCREEN subcommand.

Format **$OFFSET** or
$O

Parameters None.

Remarks

- If you have not specified the OFFSET parameter, or are in line mode, zero is returned.
- For more information, see the NOS/VE File Editor manual.

$PARAGRAPH_MARGINS EDIF Function

Purpose Returns an integer specifying the current margin setting. The keyword specified determines the value returned.

Format

$PARAGRAPH_MARGINS or
$PARAGRAPH_MARGIN or
$PM
(keyword)

Parameters

keyword

Determines the current margin for which you want a value returned. Values can be LEFT (for the left margin setting), or RIGHT (for the right margin setting), or OFFSET (for the current margin offset).

This parameter is required.

Remarks For more information, see the NOS/VE File Editor manual.

Examples The following example saves and then restores the current margin settings:

```
left_margin = $pm(left)
right_margin = $pm(right)
offset = $pm(offset)
  :
  "temporarily change the values"
  :
set_paragraph_margins mc=left_margin..right_margin ..
     o=offset
```

POSITION_BACKWARDS
EDIF Subcommand

Purpose Moves your position in the file backward a specified number of lines.

Format

POSITION_BACKWARDS or
POSB or
POSITION_BACKWARD or
PB

NUMBER=integer or *keyword*
STATUS=status variable

Parameters *NUMBER* or *N*

Specifies the number of lines to move backward. If you specify ALL, the cursor will be positioned on the first line of the file.

If omitted, 1 is assumed.

Remarks

- This subcommand is typically only used in line mode.
- For more information, see the NOS/VE File Editor manual.

Examples The following example moves the cursor backward 25 lines from the current line.

```
position_backward  number=25
```

POSITION_CURSOR
EDIF Subcommand

Purpose Locates text and positions the cursor at the specified line of text. Using this subcommand in screen mode, you can move the cursor to a nontext line.

Format

POSITION_CURSOR or
POSC or
P

TEXT=range of string
NUMBER=integer or *keyword*
LINES=range of lines or *keyword*
COLUMNS=range of integer or *keyword*
BOUNDARY=keyword
DIRECTION=keyword

UPPER_CASE=boolean
WORD=boolean
REPEAT_SEARCH=boolean
ROW=integer
STATUS=status variable

Parameters

TEXT or *T*

Specifies a text string at which to position the cursor. If omitted, the new cursor position is determined by the LINE, COLUMNS, and BOUNDARY parameters.

NUMBER or *N*

Specifies the number of times the search is to be repeated. Values can be an integer or the keyword ALL (A).

If NUMBER is omitted, and you have specified a range for the LINES parameter, ALL is assumed.

If NUMBER is omitted and no range has been specified for LINES, 1 is assumed.

LINES or *LINE* or *L*

Specifies one of two things:

- When a single line number is specified, the cursor is positioned at that line.
- When a range of lines is specified, the editor searches for the specified text string within that range of lines.

Values can be an integer, line identifier, or any of the keywords: ALL, CURRENT, FIRST, FIRST_MARK, FIRST_SCREEN, LAST, LAST_MARK, LAST_SCREEN, MARK, SCREEN.

If you omit LINE and specify BACKWARD for the DIRECTION parameter, CURRENT..FIRST is assumed.

If both LINE and DIRECTION are omitted, CURRENT..LAST is assumed.

COLUMNS or *COLUMN* or *C*

Specifies the range of columns to be searched to locate the specifed text or word. Values can be an integer from 1 through 256 or any of the keywords: CURRENT, FIRST_MARK, LAST_MARK, MARK, MAXIMUM.

When you supply a value, the BOUNDARY parameter assumes a value of STREAM.

If omitted, CURRENT is assumed. If omitted and you have specified LINE=MARK, the marked lines provide the column boundaries. If COLUMN is omitted and you have specified a LINE parameter other than MARK, all columns will be searched.

BOUNDARY or *B*

Specifies the type of boundary that will limit the search. Values can be LINE or STREAM.

If COLUMNS is specified and BOUNDARY is omitted, STREAM is assumed.

If both BOUNDARY and COLUMNS are omitted, LINE is assumed.

DIRECTION or *D*

Specifies whether to search FORWARD (F) or BACKWARD (B) from the current line.

If no value is specified, FORWARD is assumed.

UPPER_CASE or *UC*

Determines the significance of capitalization in the search.

If you specify TRUE, the editor searches the file as if it were all uppercase.

If you specify FALSE, the editor searches for the text exactly as it was entered.

If omitted, FALSE is assumed unless you specify TRUE for REPEAT_SEARCH. In this case, your last value for UPPER_CASE is used.

WORD or *W*

Determines whether the editor searches for the specified text string as a word (the text you want to search for is surrounded by nonalphanumeric characters).

If you specify TRUE, the editor searches for the text as a word. If you specify FALSE, it doesn't.

If omitted, FALSE is assumed unless you specify TRUE for REPEAT_SEARCH. In this case, your last value for WORD is used.

REPEAT_SEARCH or *RS*

Instructs the editor how to use the values entered for the last TEXT, UPPER_CASE, and WORD parameters.

If you specify TRUE, the editor uses the same TEXT, UPPER_CASE, and WORD parameters as the last time you entered them for any subcommand (unless you have specified values for this subcommand).

If you specify FALSE, the editor uses the parameters entered with the current subcommand.

If omitted, FALSE is assumed.

ROW or *R*

Enables you to move the cursor in relation to the screen instead of in relation to the file text.

Remarks For more information, see the NOS/VE File Editor manual.

Examples

- The following positions the cursor at line 500 of the file:

```
position_cursor line=500
```

- The following moves the current position backward three lines from the current line:

```
position_cursor number=3 direction=backward
```

- The following moves the cursor to the first column of the next line:

```
posc lines=current..last number=2 column=1
```

- The following moves the cursor to the second line of the current screen:

```
posc row=2
```

- The following moves the cursor to the first line in the file:

```
p line=first
```

POSITION_FORWARDS
EDIF Subcommand

Purpose Moves your position in the file forward a specified number of lines.

Format

POSITION_FORWARDS or
POSF or
POSITION_FORWARD or
PF
NUMBER=integer or *keyword*
STATUS=status variable

Parameters *NUMBER* or *N*

Specifies the number of lines to move forward. If you specify ALL, the cursor will be positioned on the last line of the file.

If omitted, 1 is assumed.

Remarks

- Using this subcommand, you cannot position past the last line of the file.
- This subcommand is typically only used in line mode.
- For more information, see the NOS/VE File Editor manual.

Examples The following example moves the cursor forward 63 lines from the current line.

```
position_forward number=63
```

PUT_ROW
EDIF Subcommand

Purpose Used with procedures to print text on any row on the screen. Enables you to display messages on different lines on the screen.

Format

PUT_ROW or
PUTR
TEXT=string
ROW=integer
STATUS=status variable

Parameters

TEXT or **T**

Specifies the text to be printed. This is a text string from 1 through 256 characters.

This parameter is required.

ROW or *R*

Indicates the row in which the text will be written. Values can be any integer from 1 through the number of rows available on your screen.

If ROW is omitted, the current line number is assumed.

Remarks

For more information, see the NOS/VE File Editor manual.

Examples

In a procedure which defines an alternate set of function key definitions for the CDC721 terminal, you might want to write the message

```
New 721 Keys are set.
```

in the message row. To do this, include the following subcommand in the procedure:

```
put_row text='New 721 keys are set.' row=$message_row
```

READ_FILE
EDIF Subcommand

Purpose

Inserts the text of another file into the current file.

Format

READ_FILE or
REAF
FILE=file
INSERTION_LOCATION=insertion_location or
keyword
PLACEMENT=keyword
MULTI_PARTITION=boolean
STATUS=status variable

Parameters

FILE or F

Specifies the name of the file from which the text is to be inserted. The entire file will be inserted. This parameter is required.

INSERTION_LOCATION or *IL*

Specifies the line before or after which the text is to be inserted (depending on the value of the PLACEMENT parameter). Values can be an integer, line identifier, or one of the LINE keywords: CURRENT, FIRST, FIRST_MARK, FIRST_SCREEN, LAST, LAST_MARK, LAST_SCREEN. Ranges are not allowed.

If omitted, CURRENT is assumed.

PLACEMENT or *P*

Specifies whether the insertion is to occur BEFORE (B) or AFTER (A) the line specified by the INSERTION_LOCATION parameter.

If omitted, AFTER is assumed.

MULTI_PARTITION or *MP*

Specifies whether the editor is to change end-of-partition delimiters to WEOP directives when an external file is copied to the current working file.

If you specify TRUE, the editor changes end-of-partition delimiters to WEOP directives and reads the entire file.

If you specify FALSE, no change takes place and the editor stops reading at the first partition.

If omitted, FALSE is assumed.

Remarks

- The READ_FILE subcommand reads the external copy of the specified file. If you have been editing a file within the editor and have not made the changes permanent using the WRITE_FILE subcommand and then specify that file on a READ_FILE subcommand, an external copy is inserted, not the changed working copy.

- For more information, see the NOS/VE File Editor manual.

Examples

- The following inserts the contents of file *ERNIE* into the current file immediately after line 320:

```
read_file file=ernie insertion_location=320
```

- The following inserts the contents of file *BERT* into the current file immediately before the last marked line:

```
reaf f=bert il=last_mark placement=before
```

REPLACE_LINES
EDIF Subcommand

Purpose Replaces lines of text by deleting the old text and replacing it with the text you specify.

Format

REPLACE_LINES or
REPLACE_LINE or
REPL
TEXT=range of string
NEW_TEXT=string
NUMBER=integer or *keyword*
LINES=range of lines or *keyword*
UNTIL=string
UPPER_CASE=boolean
WORD=boolean
REPEAT_SEARCH=boolean
STATUS=status variable

Parameters

TEXT or *T*

Specifies the text you want to replace.

If a range of text is specified, the lines containing the entire range are replaced with the string supplied in the NEW_TEXT parameter.

If TEXT is omitted, the LINE and NUMBER parameters determine the lines to be replaced.

NEW_TEXT or *NT*

Specifies the new line of text that is to replace the specified line.

If this parameter is omitted, you are prompted to enter text line by line until the editor encounters the character(s) specified by the UNTIL parameter.

NUMBER or *N*

Specifies the number of lines to replace. Values can be a number or the keyword ALL.

If a range of text is specified, NUMBER indicates the number of blocks of text to replace.

If you omit this parameter and specify a range for the LINE parameter, the assumed value is ALL.

If you omit this parameter and do not specify a range for the LINE parameter, the assumed value is 1.

LINES or *LINE* or *L*

Specifies a range of lines in which the replacement is to occur. Values can be an integer, line identifier, or one of the keywords: ALL, CURRENT, FIRST, FIRST_MARK, FIRST_SCREEN, LAST, LAST_MARK, LAST_SCREEN, MARK, SCREEN. If a single value is specified, only that line is replaced.

If omitted, CURRENT..LAST is assumed.

UNTIL or *U*

In line mode, specifies a string that stops the replacement text.

If NEW_TEXT is omitted, you are prompted to enter input until the editor encounters the character(s) you specify with the UNTIL parameter.

If the UNTIL parameter is omitted, ** is assumed.

UPPER_CASE or *UC*

Determines the significance of capitalization in the search.

If you specify TRUE, the editor searches the file as if it were all uppercase.

If you specify FALSE, the editor searches for the text exactly as it was entered.

If omitted, FALSE is assumed unless you specify TRUE for REPEAT_SEARCH. In this case, your last value for UPPER_CASE is used.

WORD or *W*

Determines whether the editor searches for the specified text string as a word (the text you want to search for is surrounded by nonalphanumeric characters).

If you specify TRUE, the editor searches for the text as a word. If you specify FALSE, it doesn't.

If omitted, FALSE is assumed unless you specify TRUE for REPEAT_SEARCH. In this case, your last value for WORD is used.

REPEAT_SEARCH or *RS*

Instructs the editor how to use the values entered for the last TEXT, UPPER_CASE, and WORD parameters.

If you specify TRUE, the editor uses the same TEXT, UPPER_CASE, and WORD parameters as the last time you entered them for any subcommand (unless you have specified values for this subcommand).

If you specify FALSE, the editor uses the parameters entered with the current subcommand.

If omitted, FALSE is assumed.

Remarks For more information, see the NOS/VE File Editor manual.

Examples

- The following replaces lines 30 to the end of the file with a line that says *text*:

```
replace_line new_text='text' line=30..last
```

- The following replaces the current line with text you are prompted to enter until the editor encounters ** at the end of one of the replacement lines.

```
repl
```

REPLACE_TEXT EDIF Subcommand

Purpose Replaces blocks of text.

Format

REPLACE_TEXT or
REPT or
R

TEXT=string
NEW_TEXT=string
NUMBER=integer or *keyword*
LINES=range of lines or *keyword*
COLUMNS=range of integer or *keyword*
BOUNDARY=keyword

UPPER_CASE=boolean
WORD=boolean
REPEAT_SEARCH=boolean
VETO=boolean
STATUS=status variable

Parameters

TEXT or *T*

Specifies the text string to replace in the specified block of text.

If omitted, REPEAT_SEARCH is required.

NEW_TEXT or *NT*

Specifies the replacement text for the string specified in the TEXT parameter.

If omitted, the string specified in the TEXT parameter is deleted.

NUMBER or *N*

Specifies the number of times the original text is to be replaced within the block of text. Values can be any integer or the keyword ALL (A).

If you omit this parameter and specify a range of values for the LINE parameter, the assumed value is ALL.

If you omit this parameter and do not specify a range for the LINE parameter, the assumed value is 1.

LINES or *LINE* or *L*

Specifies the range of lines affected by the replacement. Values can be an integer, line identifier, or one of the keywords: ALL, CURRENT, FIRST, FIRST_MARK, FIRST_SCREEN, LAST, LAST_MARK, LAST_SCREEN, MARK, SCREEN. If a single value is specified, only the number of occurrences of text are replaced in that line.

If omitted, CURRENT..LAST is assumed.

COLUMNS or *COLUMN* or *C*

Specifies the range of columns affected by the replacement. The integers can be from 1 through 256 or any of the keywords: CURRENT, FIRST_MARK, LAST_MARK, MARK, MAXIMUM.

With the COLUMN parameter you can specify a beginning and ending column for the replacement. When you specify a boundary of STREAM, the search for

replacement starts at the beginning column on the beginning line, continues through all columns of the next lines, and stops at the end column of the ending line.

If COLUMN, BOUNDARY, and LINE are omitted and NUMBER=ALL, the replacement search starts at the current column of the current line and ends at the last column of the last line. If COLUMN is omitted and LINE is specified, the replacement search uses all columns of the lines specified. If COLUMN, BOUNDARY, LINE, and NUMBER are omitted, the current column is assumed.

BOUNDARY or *B*

Specifies the type of boundary that will limit the replacement. Values can be LINE or STREAM.

If BOUNDARY and COLUMNS are both omitted, LINE is assumed.

If BOUNDARY is omitted but COLUMNS is specified, STREAM is assumed.

UPPER_CASE or *UC*

Determines the significance of capitalization in the search.

If you specify TRUE, the editor searches the file as if it were all uppercase.

If you specify FALSE, the editor searches for the text exactly as it was entered.

If omitted, FALSE is assumed unless you specify TRUE for REPEAT_SEARCH. In this case, your last value for UPPER_CASE is used.

WORD or *W*

Determines whether the editor searches for the specified text string as a word (the text you want to search for is surrounded by nonalphanumeric characters).

If you specify TRUE, the editor searches for the text as a word. If you specify FALSE, it doesn't.

If omitted, FALSE is assumed unless you specify TRUE for REPEAT_SEARCH. In this case, your last value for WORD is used.

REPEAT_SEARCH or *RS*

Instructs the editor how to use the values entered for the last TEXT, UPPER_CASE, and WORD parameters.

If you specify TRUE, the editor uses the same TEXT, NEW_TEXT, UPPER_CASE, and WORD parameters as the last time you entered them for any subcommand (unless you have specified values for this subcommand).

If you specify FALSE, the editor uses the parameters entered with the current subcommand.

If omitted, FALSE is assumed.

VETO or *V*

Enables you to display a directory of replaced lines allowing you to choose a line at which you want the cursor to be positioned. Allows you to veto any of the displayed lines affected by the subcommand.

Remarks

- If you want to replace text within certain columns in many lines as in the following example,

Before replacement:	After replacement:
rrrrrrr	rrrtttr
rrrrrrr	rrrtttr
rrrrrrr	rrrtttr
rrrrrrr	rrrtttr

  you should not use the COLUMNS parameter for this subcommand. Rather, use the SET_SEARCH_MARGINS subcommand followed by the REPLACE_TEXT subcommand.

- For more information, see the NOS/VE File Editor manual.

Examples

- The following changes the first occurrence of *water* to *wine* from the current line and column to the last line and column:

```
replace_text text='water' new_text='wine'
```

- The following uses the same values for TEXT, NEW_TEXT, UPPER_CASE, and WORD parameters specified on a previous subcommand:

```
rept repeat_search=true
```

- The following replaces all occurrences of *JILL* with *BETTY* from line 50 to the end of the file:

```
r text='JILL' new_text='BETTY' line=50..last
```

- The following replaces the first occurrence of *$* with *#* from the current line and column to the last line and column:

```
rept t='$' nt='#'
```

- The following deletes the text *bye* in all lines of the file:

```
r text='bye' line=a
```

- The following changes *r* to *t* starting at line 2 column 5, and ending at line 4 column 3.

```
replace_text 'r' 't' line=2..4 column=5..3
```

The following occurs:

Before replacement:	After replacement:
rrrrrrr	rrrrrrr
rrrrrrr	rrrrttt
rrrrrrr	ttttttt
rrrrrrr	tttrrrr

RESET_FILE EDIF Subcommand

Purpose Cancels all the changes you have made to your current file since you last accessed the file using the EDIT_FILE command.

Format

RESET_FILE or
RESF
STATUS=status variable

Remarks

- The RESET_DECK subcommand discards changes to decks.
- For more information, see the NOS/VE File Editor manual.

RESTORE_POSITION
EDIF Subcommand

Purpose Enables you to return to the position saved by the SAVE_POSITION subcommand.

Format

RESTORE_POSITION or
RESP
STATUS=status variable

Remarks For more information, see the NOS/VE File Editor manual.

$ROW_TEXT
EDIF Function

Remarks Reserved for site personnel, Control Data, or future use.

SAVE_POSITION
EDIF Subcommand

Purpose Enables you to save the current column, line, and file name for reference later.

Format

SAVE_POSITION or
SAVP
STATUS=status variable

Remarks

- To return to this position later, use the RESTORE_POSITION subcommand.
- For more information, see the NOS/VE File Editor manual.

$SCREEN_ACTIVE
EDIF Function

Purpose Returns a boolean value. It is TRUE if screen mode is active, and FALSE if it is not.

Format

$SCREEN_ACTIVE or
$SA

Parameters None.

Remarks For more information, see the NOS/VE File Editor manual.

$SCREEN_INPUT EDIF Function

Purpose Returns the text you enter on the subcommand line as the string for the TEXT parameter value.

Format

$SCREEN_INPUT or
$SI
(*string*)

Parameters *string*

The text you want displayed on the message row as a prompt for input. If omitted, ENTER TEXT is used as the prompt.

Remarks

- This function allows an SCL procedure to pause and request input.
- When the LOCATE_TEXT subcommand is executed, the user provides the text normally, without concern that it will become a string. The user does not put apostrophes around the text or use double apostrophes within the text.
- For more information, see the NOS/VE File Editor manual.

Examples

- The following subcommand locates whatever text the user provides in response to $SI.

```
locate_text t=$si('What do you want to locate?')
```

- The following subcommand programs key 7 to insert whatever characters are specified:

```
setfk n=7 ..
   cs='insc '//$quote($SI('characters ..
   to be inserted by key 7'))
```

$SEARCH_MARGINS
EDIF Function

Purpose Returns an integer specifying the column number of either the right or left margin. The keyword specified determines the value returned.

Format

$SEARCH_MARGINS or
$SEARCH_MARGIN or
$SM
(keyword)

Parameters **keyword**

Specifies the margin for which you want a value returned. Values can be LOW (for the left margin) or HIGH (for the right margin).

This parameter is required.

Remarks

- The function can be used to save the values for the current search margins so they can be temporarily altered.
- For more information, see the NOS/VE File Editor manual.

SET_EPILOG
EDIF Subcommand

Purpose Specifies a file containing editor subcommands you want executed each time you leave the editor.

Format

SET_EPILOG or
SETE
FILE=file
STATUS=status variable

Parameters *FILE* or *F*

Specifies the file to contain the editor subcommands. If omitted, $USER.SCU_EDITOR_EPILOG is assumed.

Remarks

- If you do not enter a SET_EPILOG subcommand within your editing session, no epilog file is executed.
- You can enter this command anytime within your editing session.
- If you want epilog file processing to occur automatically, put the SET_EPILOG subcommand into your prolog file.
- For more information, see the NOS/VE File Editor manual.

Examples

The following process always leaves your screen display at 132 columns after stopping the editor:

1. Place the following in file $USER.SCU_EDITOR_EPILOG.

```
if $screen_active then;setso c=132;ifend
```

2. Include the following subcommand in your prolog file.

```
set_epilog
```

SET_FUNCTION_KEY EDIF Subcommand

Purpose

Enables you to create your own set or sets of function keys.

Format

SET_FUNCTION_KEY or
SETFK
NUMBER=integer or **keyword**
COMMAND_STRING=string
SHIFT=boolean
LABEL=string
STATUS=status variable

Parameters

NUMBER or **N**

Specifies the number of the key to be defined. Values can be any integer from 1 through 16. These numbers correspond to the highlighted boxes in the menu of operations at the bottom of the screen. The numbers 1 through 8 correspond to the first row of boxes; 9 through 16 correspond to the second row of boxes.

You can also specify one of the following keywords: DOWN(D), EDIT(E), FWD(F), BKW(B), BACK, HELP(H) STOP(S), UNDO, UP(U).

The keywords relate to keys on some terminals. If your terminal has defined sequences that relate to these keywords, you can create your own function keys using these keywords.

This parameter is required.

COMMAND_STRING or **CS**

Specifies the subcommand(s) to be executed when the specified key is pressed. Values can be any editor or SCL command. When more than one subcommand is specified, separate them with semicolons.

This parameter is required.

SHIFT or *S*

For those terminals that have one key identifier next to each highlighted box in the menu of operations, the SHIFT parameter indicates whether the key to be used is shifted. Specify TRUE for the shifted key and FALSE for the nonshifted key.

For those terminals that have two key identifiers next to each highlighted box in the menu of operations, the SHIFT parameter indicates which key you use.

Specify TRUE to use the key corresponding to the top key identifier. Specify FALSE to use the key corresponding to the bottom key identifier.

If SHIFT is omitted, FALSE is assumed.

LABEL or *L*

Specifies a string as the label that is to appear in the menu of operations for the specified key.

If LABEL is omitted, current label remains the same.

Remarks For more information, see the NOS/VE File Editor manual.

Examples

- The following SET_FUNCTION_KEY subcommand defines the shifted F5 key to execute the HELP subcommand. The key has a screen label of help:

```
set_function_key number=5 command_string='help' ..
   shift=true label='help'
```

- The online Examples manual lists a number of useful function key definitions.

SET_LINE_WIDTH EDIF Subcommand

Purpose Specifies the maximum line length. When a line exceeds this limit, a warning message is displayed.

Format

SET_LINE_WIDTH or
SETLW
WIDTH=integer
STATUS=status variable

Parameters

WIDTH or **W**

Specifies the number of characters you can have on one line before the editor sends you a message. Values can be an integer from 0 through 256. Specifying 0 eliminates the message and adds no trailing blanks to lines. When you create a file, an initial width value of 0 is assumed. For decks the value is taken from the deck header information.

This parameter is required.

Remarks

- Each time you edit a file, you must enter the SET_LINE_WIDTH subcommand to be warned when lines exceed a given length.
- Once this command is entered, the editor adds trailing spaces to lines with a character count less than the limit for string comparisons.
- You can locate long lines using the LOCATE_WIDE_LINES subcommand.
- For more information, see the NOS/VE File Editor manual.

Examples The following subcommand sets the line width limit at 80:

```
set_line_width width=80
```

SET_LIST_OPTIONS
EDIF Subcommand

Purpose Provides you with the options in line mode of either displaying the line identifier on the same line as the text, on a separate line from the text, or not at all.

Format

SET_LIST_OPTIONS or
SET_LIST_OPTION or
SETLO
LINE_IDENTIFIER=keyword
STATE=boolean
STATUS=status variable

Parameters *LINE_IDENTIFIER* or *LI*

Specifies where or if the identifier is to be displayed. Values can be LEFT (L), SEPARATE (S), or NONE.

If LINE_IDENTIFIER is omitted, NONE is assumed.

STATE or *S*

Specifies whether the state of the modification associated with the line's introduction is to be displayed.

If TRUE, the state is displayed.

If omitted, FALSE is assumed.

Remarks

- This subcommand is usually entered when you are line editing decks and want to see the line identifiers.
- Modification states are described in the NOS/VE Source Code Management manual.
- For more information, see the NOS/VE File Editor manual.

SET_MASK
EDIF Subcommand

Purpose When specifying a value for the TEXT parameter, you can specify a special character that can be used to match any other character. This character serves as a wild card character.

Format

SET_MASK or
SETM
CHARACTER=string or **keyword**
STATUS=status variable

Parameters

CHARACTER or **C**

Specifies the mask character. Values can be any alphanumeric character or the keyword NONE. If NONE is specified, the mask feature is turned off.

This parameter is required.

Remarks

- When you start editing, no mask character is set.
- For more information, see the NOS/VE File Editor manual.

Examples

- The following changes the SET_MASK character to #:

```
set_mask character=#
```

- The following are strings that match the string *'F##d'*:

```
Ford Fred Food Find Fund
```

SET_PARAGRAPH_MARGINS EDIF Subcommand

Purpose

Changes the paragraph margins. In any subsequent FORMAT_PARAGRAPH or CENTER_LINE subcommands, the margins set with SET_PARAGRAPH_MARGINS are used.

Format

SET_PARAGRAPH_MARGINS or
SET_PARAGRAPH_MARGIN or
SETPM
MARGIN_COLUMNS=range of integer
OFFSET=integer
STATUS=status variable

Parameters

MARGIN_COLUMNS or *MARGIN_COLUMN* or *MC*

Specifies the left and right margins. If just one column number is specified, the left margin is set to that number.

If omitted and you have not specified this subcommand previously in your editing session, columns 1 and 65 are used. If you have specified the subcommand previously, any parameter not specified is not changed.

OFFSET or *O*

Specifies the number of columns the first line in the paragraph is to be offset from the rest of the lines in the paragraph. If the number specified is a positive number, the first line of the paragraph is indented the number of columns specified. If zero is specified, the first line is not indented. If a negative value is given, the first line begins to the left of the rest of the paragraph.

If omitted and you have specified this subcommand during this terminal session, the previous value is used. If you have not entered this subcommand previously and omit the OFFSET parameter, 4 is assumed.

Remarks

- You can use the $PARAGRAPH_MARGINS function to return paragraph margin values.
- For more information, see the NOS/VE File Editor manual.

Examples

- To set the paragraph margins to columns 7 and 72, with an offset of 4, enter:

```
set_paragraph_margins margin_columns=7..72
```

- To set the margins to 10 and 70 and also specify that you want the first line of the paragraph indented 5 columns, enter:

```
setpm mc=10..70 o=5
```

SET_SCREEN_OPTIONS EDIF Subcommand

Purpose Enables you to change the way the screen appears. Among other things, you can change the number of lines that are listed on your screen, the number of rows in the menu of operations that is displayed, the number of files displayed at one time, and the number of columns displayed.

Format

SET_SCREEN_OPTIONS or
SET_SCREEN_OPTION or
SETSO
MODEL=name
COLUMNS=integer
MENU_ROWS=integer
ROWS=integer
SPLITS=integer
SPLIT_SIZES=list of integer
STATUS=status variable

Parameters *MODEL* or *M*

Specifies the type of terminal you are using. Valid entries are:

Entry	Terminal
CDC_721	Control Data 721
CDC_722	Control Data 722
CDC_722_30	Control Data 722-30
MAC_CONNECT_10	Apple Macintosh running version 1.0 or 1.0+ of Control Data CONNECT for the Macintosh
MAC_CONNECT_11	Apple Macintosh running version 1.1 of Control Data CONNECT for the Macintosh
PC_CONNECT_10	IBM PC or equivalent running version 1.0 of Control Data CONNECT for the IBM PC

PC_CONNECT_11	IBM PC or equivalent running version 1.1 of Control Data CONNECT for the IBM PC
PC_CONNECT_12	IBM PC or equivalent running version 1.2 of Control Data CONNECT for the IBM PC
PC_CONNECT_13	IBM PC or equivalent running version 1.3 of Control Data CONNECT for the IBM PC
DEC_VT100_GOLD	Digital Equipment VT100
DEC_VT220	Digital Equipment VT220
ZEN_Z19	Zenith Z19 or Heathkit H19
ZEN_Z29	Zenith Z29

If the MODEL parameter has not been specified on an earlier subcommand of the editing session, or by a CHANGE_TERMINAL_ATTRIBUTES TM=name command previous to the editing session, it is required.

COLUMNS or *COLUMN* or *C*

Specifies the number of columns to be displayed. Values range from 1 to the maximum number allowed on your terminal. The number you enter is compared to the screen sizes set up in the terminal definition for your terminal. The number of columns displayed is the closest number as large or larger than the number you enter on the COLUMNS parameter.

Each time the editor is entered, a value of 80 columns is assumed.

If COLUMN is omitted, the number of columns displayed remains the same.

MENU_ROWS or *MENU_ROW* or *MR*

Specifies the number of rows of the menu of operations prompts to display. Values can be:

0 Does not display the menu of operations.

1 Displays 1 row of highlighted boxes from the menu of operations.

2 Displays 2 rows of the menu.

If MENU_ROW is omitted, the number of rows displayed remains the same. When starting the editor, 1 row is displayed.

ROWS or *ROW* or *R*

Specifies the number of rows to display for terminals that support multiple screen sizes. Values can be from 10 to the maximum number allowed for your terminal. The number you enter is compared to the screen sizes set up in the terminal definition for your terminal. The number of rows displayed is the closest number as large or larger than the number you enter on the ROWS parameter.

When you first enter the editor, it assumes a value of 32.

Not all terminals support multiple screen sizes.

SPLITS or *SPLIT* or *S*

Specifies the number of areas of text (splits) you want displayed on the screen when the screen is divided horizontally to show more than one file. This number determines how many files you can display at the same time. Values are 1 through 16.

Each time the editor is entered, a value of 1 is assumed.

If SPLIT is omitted, the number of splits remains the same.

SPLIT_SIZES or *SPLIT_SIZE* or *SS*

Specifies the number of lines you want displayed within a particular area of text (split). The value(s) you specify correspond positionally to the splits displayed; the first value you specify corresponds to the topmost split, the second value to the next lowest split and so on. Values are 2 through 255.

If SPLIT_SIZE is omitted, each split contains an equal number of lines.

Remarks

- For all omitted parameters, the editor assumes you want the same value used the last time you entered the SET_SCREEN_OPTIONS subcommand.
- For more information, see the NOS/VE File Editor manual.

Examples

- The following example displays an additional file onto a screen. The new screen contains two split areas with a different file in each area.

  1. Press **Home** and enter:

     ```
     set_screen_options split=2
     ```

  2. Move the cursor to the split in which you want the new file (ZAP) to appear.

  3. Press **Home** and enter:

     ```
     edif zap
     ```

     File ZAP appears in the split area the cursor was last in.

- The following example displays all of your menu of operations:

  ```
  setso menu_row=2
  ```

SET_SEARCH_MARGINS
EDIF Subcommand

Purpose Limits the number of columns to be searched in subsequent subcommands that use string searches.

Format

SET_SEARCH_MARGINS or
SET_SEARCH_MARGIN or
SETSM
MARGIN_COLUMNS=range of integer
STATUS=status variable

Parameters *MARGIN_COLUMNS* or *MC*

Specifies the column(s) in which to perform the search. Values can be any number or any of the COLUMN keywords: CURRENT, FIRST_MARK, LAST_MARK, MARK, MAXIMUM. If you specify two values, the search is done from the first column through the last column specified. If you specify a single integer, only that column is searched.

If MARGIN_COLUMN is omitted, columns 1 through 256 are assumed.

Remarks

- The $SEARCH_MARGINS function can be used to return the MARGIN_COLUMNS values.
- This subcommand can be used with the REPLACE_TEXT subcommand to change a string within a limited range of columns for many lines.
- For more information, see the NOS/VE File Editor manual.

Examples To set the search margins to columns 1 and 7, enter:

```
set_search_margins margin_columns=1..7
```

SET_TAB_OPTIONS EDIF Subcommand

Purpose Sets a tab character and the columns in which you want tabs set.

Format

SET_TAB_OPTIONS or
SET_TAB_OPTION or
SETTO
CHARACTER=string
TAB_COLUMN=list of integer
STATUS=status variable

Parameters *CHARACTER* or *C*

Specifies the tab character. Values can be any character. The horizontal tab character, $char(9), works well as a value.

When you enter a tab character within text typed from your terminal, the tab character moves any text from the current position to the next tab setting.

If you enter a tab character after the last tab column, the tab character is included as part of the file text.

When you start editing a file, the tab character is set to the reverse slant. When you start editing a deck, the tab character is set as specified in the deck header (refer to the CREATE_DECK SCU subcommand in the NOS/VE Source Code Management manual).

If CHARACTER is omitted, the tab character is not changed.

TAB_COLUMN or *TAB_COLUMNS* or *TC*

Specifies tab columns to be added to those already selected.

A maximum of 256 columns can be specified as tab columns. Values can be any integer from 1 through 256 and must be enclosed in parentheses. When you start editing a file, the tabs are set at columns 1, 7, and 72.

When you start editing a deck, the tab columns selected are those specified in the deck header (refer to the CREATE_DECK SCU subcommand in the NOS/VE Source Code Management manual).

If TAB_COLUMN is omitted, the tab settings are not changed.

Remarks For more information, see the NOS/VE File Editor manual.

Examples

- The following sets the tab character to] and adds columns 11, 18, 41 and 53 as tab columns:

```
set_tab_options character=']' ..
    tab_column=(11,18,41,53)
```

- The following sets the tab character to ! and adds column 3 as a tab column:

```
set_tab_options character='!' tab_column=(3)
```

SET_VERIFY_OPTION EDIF Subcommand

Purpose Displays lines that have been changed using the REPLACE_TEXT subcommand and displays the first and last lines of a block of text located with the LOCATE_TEXT subcommand.

Format

SET_VERIFY_OPTION or
SETVO
ECHO=boolean
STATUS=status variable

Parameters

ECHO or **E**

Specifies whether you want the verify option on or off.

This parameter is required.

Remarks

- In screen mode the verify option is always off.
- The system sets the verify option to TRUE when you start the editor. Therefore, in line mode the verify option is on unless you specify ECHO=FALSE.
- The function $VERIFY_OPTION returns the current value of the verify option.
- For more information, see the NOS/VE File Editor manual.

SET_WORD_CHARACTERS
EDIF Subcommand

Purpose

Enables you to add or delete allowable characters (within words) for use with the WORD parameter.

Format

SET_WORD_CHARACTERS or
SET_WORD_CHARACTER or
SETWC
ADD=list of string
DELETE=list of string
STATUS=status variable

Parameters

ADD or *A*

Specifies the characters to add as allowable characters. Values can be any printable character. The space character cannot be specified as an allowable character.

Enclose each character in quotes and all inside parentheses.

If ADD is omitted, no characters are added.

DELETE or *D*

Specifies the characters to delete as allowable characters in a word. In other words, characters specified by this parameter will be treated as punctuation marks. Values can be any printable character. The space character is not allowed.

Enclose each character in quotes and all inside parentheses.

If DELETE is omitted, no characters are deleted.

Remarks

- The initial word characters consist of the alphanumerics plus the underscore (_), dollar-sign ($), number-sign (#), and at-sign (@).
- If you specify more than one character, separate them with commas or spaces.
- For more information, see the NOS/VE File Editor manual.

Examples

- The following adds % as an allowable word character and deletes x as an allowable word character:

```
set_word_characters add='%' delete='x'
```

- The following changes the characters allowed in words to those used in the NOS/VE COBOL compiler:

```
setwc add='-' delete('$' '#' '_' '@')
```

$SPLIT_SIZE
EDIF Function

Purpose Returns an integer specifying the number of available text lines for the specified split of the screen.

Format

$SPLIT_SIZE or
$SS
(*integer*)

Parameters *integer*

Specifies the split of the screen for which you want a value returned. If omitted, the current split is assumed.

Remarks

- If you are in line mode, zero is returned.
- For more information, see the NOS/VE File Editor manual.

$TEXT
EDIF Function

Purpose Returns a string specifying the last text you specified for a TEXT parameter.

Format

$TEXT or
$T

Parameters None.

Remarks For more information, see the NOS/VE File Editor manual.

$TITLE_ROW
EDIF Function

Purpose Returns an integer specifying the row number of the title row (file header) used for the specified split of the screen.

Format

$TITLE_ROW or
$TR
(*integer*)

Parameters *integer*

Specifies the split of the screen for which you want a value returned. If omitted, the current split is assumed.

Remarks

- If you are in line mode, zero is returned.
- For more information, see the NOS/VE File Editor manual.

UNDO
EDIF Subcommand

Purpose Cancels changes in reverse chronological order. Entire transactions are undone until one is undone that included a change in text.

Format

UNDO or
UND
STATUS=status variable

Remarks

- A transaction consists of all changes made between two presses of the return key.
- The following terminals include an automatic return when you press keys that perform editing operations:

  CDC 721
  CDC 722-30
  IBM PC
  Apple Macintosh

  At these terminals, pressing keys that perform editing operations marks the end of a transaction. At other terminals, you press return to end transactions that include editing operations.
- Use the UNMARK subcommand to cancel marks.
- For each UNDO subcommand, all changes made since the last time you pressed the return key are canceled.
- You can undo only changes made to the current file. You can, however, make any file that was edited during this session the current file if it has not been closed with END_FILE, END_DECK, or a SELECT_DECK subcommand. You can do this by entering the EDIT_FILE or EDIT_DECK subcommand, or, if your screen is split, by positioning the cursor in the file you want to be the current file.
- To undo all changes you have made since opening the current file, use the RESET_FILE subcommand.
- For more information, see the NOS/VE File Editor manual.

Examples

The following changes were made to a file in the order given:

1. Five lines in the file were deleted using one DELETE_LINES subcommand.
2. The next three lines are displayed using the LOCATE_TEXT subcommand.

3. A new line is entered using the INSERT_LINES subcommand.

Each time UNDO is entered, the following changes are undone:

1. The first time UNDO is entered, the new line inserted is deleted.

2. The second time, the five lines deleted are returned.

UNMARK
EDIF Subcommand

Purpose Explicitly cancels the marks on any lines or characters you previously marked.

Format

UNMARK or
UNM
STATUS=status variable

Remarks

- You implicitly unmark text by marking a new region of text, by deleting marked text, or by entering the Undo operation (or UNDO subcommand), which undoes the most recent change as well as undoing any current marks.

- When you enter the END_FILE subcommand you can close a file containing the marked text.

- For more information, see the NOS/VE File Editor manual.

$UPPER_CASE
EDIF Function

Purpose Returns a boolean value specifying the most recent value supplied for an UPPER_CASE parameter.

Format

$UPPER_CASE or
$UC

Parameters None.

Remarks For more information, see the NOS/VE File Editor manual.

$VERIFY_OPTION
EDIF Function

Purpose Returns a boolean value indicating whether the VERIFY option has been activated (TRUE) or not (FALSE).

Format **$VERIFY_OPTION** or
$VO

Parameters None.

Remarks For more information, see the NOS/VE File Editor manual.

$WORD
EDIF Function

Purpose Returns a boolean value indicating whether the word search feature is active (TRUE) or not (FALSE).

Format **$WORD** or
$W

Parameters None.

Remarks For more information, see the NOS/VE File Editor manual.

WRITE_FILE
EDIF Subcommand

Purpose Copies text from the current working file to the external copy of a file.

Format **WRITE_FILE** or
WRIF
TEXT=range of string
NUMBER=integer or *keyword*
LINES=range of lines or *keyword*
FILE=file
UPPER_CASE=boolean
WORD=boolean
REPEAT_SEARCH=boolean
MULTI_PARTITION=boolean
STATUS=status variable

Parameters *TEXT* or *T*

Specifies string(s) of text in the first and last lines of a block of text to be written.

If you enter only one string, the block of text to be written will contain only one line. If you enter two strings, the search for the second begins immediately after the first is found and the cursor is positioned at the beginning of the first string.

If omitted, the lines to be written are determined by the NUMBER, LINE, and DIRECTION parameters or by the REPEAT_SEARCH parameter.

NUMBER or *N*

Specifies the number of blocks of text to be copied. Values for this parameter can be an integer the keyword ALL (A).

If NUMBER is omitted, ALL is assumed.

LINES or *LINE* or *L*

Specifies a range of lines to be searched to locate the text to be copied. Values can be an integer, line identifier, or one of the keywords: ALL, CURRENT, FIRST, FIRST_MARK, FIRST_SCREEN, LAST, LAST_MARK, LAST_SCREEN, MARK, SCREEN.

If a single value is specified, only that line is searched.

If LINE is omitted, ALL is assumed.

FILE or *F*

Specifies the file to which the text is to be copied.

If the object you are editing is a file and FILE is omitted, the editor writes the file to the external file from which the working file was made.

If the object you are editing is a deck, this parameter is required.

UPPER_CASE or *UC*

Determines the significance of capitalization in the search.

If you specify TRUE, the editor searches the file as if it were all uppercase.

If you specify FALSE, the editor searches for the text exactly as it was entered.

If omitted, FALSE is assumed unless you specify TRUE for REPEAT_SEARCH. In this case, your last value for UPPER_CASE is used.

WORD or *W*

Determines whether the editor searches for the specified text string as a word (the text you want to search for is surrounded by nonalphanumeric characters).

If you specify TRUE, the editor searches for the text as a word. If you specify FALSE, it doesn't.

If omitted, FALSE is assumed unless you specify TRUE for REPEAT_SEARCH. In this case, your last value for WORD is used.

REPEAT_SEARCH or *RS*

Instructs the editor how to use the values entered for the last TEXT, UPPER_CASE, and WORD parameters.

If you specify TRUE, the editor uses the same TEXT, UPPER_CASE, and WORD parameters as the last time you entered them for any subcommand (unless you have specified values for this subcommand).

If you specify FALSE, the editor uses the parameters entered with the current subcommand.

If omitted, FALSE is assumed.

MULTI_PARTITION or *MP*

Specifies whether the editor is to change WEOP directives to end-of-partition delimiters when the current working file is copied to an external file.

If TRUE, the editor changes WEOP directives to end-of partition delimiters.

If FALSE, no substitution takes place.

If omitted, FALSE is assumed.

Remarks For more information, see the NOS/VE File Editor manual.

Examples

- The following copies 3 blocks of text beginning with the line containing *even* and ending with the line containing *odd* to the file BOTH:

```
write_file text='even'..'odd' number=3 file=both
```

- The following copies all lines from the current file to the external copy of file SPLAT:

```
write_file line=all file=splat
```

- The following copies all of the current working file to the end of file ZAP:

```
wrif file=ZAP.$EOI
```

- The following copies the working copy of the current file to the external copy. In other words, it makes your changes permanent without closing the current file and leaving the editor:

```
wrif
```

MANAGE_REMOTE_FILES 16

MANAGE_REMOTE_FILE Command

Purpose Delimits a set of commands to be executed on the specified remote system.

Format

MANAGE_REMOTE_FILE or
MANAGE_REMOTE_FILES or
MANRF or
MFLINK
LOCATION=any
FILE=file
DATA_DECLARATION=keyword
UNTIL=string
SUBSTITUTION_MARK=string or *keyword*
STATUS=status variable

Parameters

LOCATION or **L**

Specifies the name of the remote location to be accessed. This is a name associated with a remote system, such as a family name or a logical identifier. (Location names are determined by your network application administrator.)

You cannot specify a variable name for this parameter. If you want to use a variable that has a name value, you can use the $NAME function instead.

This parameter is required.

FILE or **F**

Specifies the name of a file on the local NOS/VE system to be used as the input or output file during a file transfer. This parameter is required even when you are not performing a file transfer.

DATA_DECLARATION or *DD*

Specifies the data format of a file to be transferred.

If the remote location is another NOS/VE host, this parameter is ignored. The rules for copying NOS/VE files based on the local and remote file attributes apply. For a discussion of rules for copying NOS/VE files, see the NOS/VE System Usage manual.

If the remote location is a non-NOS/VE host, the following data descriptions are available. The meaning of each varies among the various remote host types. Refer to the Remote Host Facility Usage manual for system specific information

- C6

  Use this format when you transfer files to hosts using a six-bit code set. This format indicates the file contains character data from a character set with 64 or fewer character codes.

  The effect of this format is that that each machine sees the file in its native character set. Thus, if you transfer the file from NOS/VE to NOS, NOS/VE sends the file in ASCII and NOS receives it in display code. Transfers to other systems result in full ASCII transfers as if DD=C8 was used.

- C8

  This format has the following meanings depending upon which system the file is being transfered to:

  - NOS

    Transfer results in a NOS 8/12 ASCII file. Use the NOS FCOPY command to convert the file to NOS 6/12 format.

  - NOS/BE

    Same as for NOS.

  - Any other ASCII system

    Transfer results in an ASCII file.

  - IBM/MVS

    Transfer results in an EBCDIC file.

- UU

  Use this format to transfer binary files to remote systems. Object and source libraries should be transferred using this format. Files transferred to NOS or NOS/BE will be padded unless they end on a 120 bit boundry (this is because NOS and NOS/BE store their files in 60 bit format). Similarily, files transferred from NOS or NOS/BE

to NOS/VE and that have a file length that is an odd multiple of 60 bits will be padded to the next full byte (8 bit) length.

UNTIL or *U*

Specifies the string indicating the end of commands in the list. The string must appear on a separate line. If this parameter is omitted, a string of two asterisks (**) is assumed.

SUBSTITUTION_MARK or *SM*

Specifies a character used to delimit text to be substituted within the command text following the MANAGE_REMOTE_FILES command. Values can be any character or the keyword NONE. NONE specifies that no substitution mark is to be used. If this parameter is omitted, NONE is assumed.

Remarks

- You must provide validation information required by the remote system. If this remote system is NOS/VE, the first command in the list of commands must be a LOGIN command. Alternately, you can issue a CREATE_REMOTE_VALIDATION command prior to using the MANAGE_REMOTE_FILES command.

- The names and parameters of commands accepted by each remote system type are described in the Remote Host Facility Usage manual.

  The MANAGE_REMOTE_FILES command passes the command text you supply to the remote system for execution. If the remote system is NOS/VE, the command text is a set of SCL commands to be executed as a batch job.

- You can include at most one remote command in the command text which causes an explicit file transfer. For remote NOS/VE systems, use the SEND_FILE or RECEIVE_FILE commands to explicitly transfer a file.

- For more information, see the NOS/VE System Usage manual.

RECEIVE_FILE
MANRF Subcommand

Purpose When used within the list of commands delimited by the MANAGE_REMOTE_FILES command, transfers a file from your local system to a remote system.

Format

RECEIVE_FILE or
RECF
FILE=file
STATUS=status variable

Parameters

FILE or **F**

Specifies the name of the file on the remote system that is to receive the file from your local system.

Remarks

- You can use the RECEIVE_FILE command only with the MANAGE_REMOTE_FILES command. (Refer to the MANAGE_REMOTE_FILES command.)
- Refer to the SEND_FILE command for informtion about transferring files from a remote system to your local system.
- For more information, see the NOS/VE System Usage manual.

SEND_FILE
MANRF Subcommand

Purpose When used within the list of commands delimited by the MANAGE_REMOTE_FILES command, sends a file from a remote system to your local system.

Format

SEND_FILE or
SENF
FILE=file
STATUS=status variable

Parameters

FILE or **F**

Specifies the name of the file on the remote system that is to be sent to your local system.

Remarks

- You can use the SEND_FILE command only with the MANAGE_REMOTE_FILES command. (Refer to the MANAGE_REMOTE_FILES command.)
- Refer to the RECEIVE_FILE command for informtion about transferring files from your local system to a remote system.
- For more information, see the NOS/VE System Usage manual.

MEASURE_PROGRAM_EXECUTION 17

MEASURE_PROGRAM_EXECUTION Command

Purpose Starts a program measurement utility session.

Format

MEASURE_PROGRAM_EXECUTION or
MEAPE
STATUS=status variable

Remarks

- The session ends when you enter the subcommand QUIT. The descriptions of each program measurement subcommand follow this description.
- For more information, see the NOS/VE Object Code Management manual.

Examples The following utility session specifies the program as the modules on file LGO, executes the program, and saves the program profile on file MY_FILE.

```
/measure_program_execution
MPE/set_program_description target_text=lgo
MPE/execute_instrumented_task
MPE/display_program_profile output=my_file
MPE/quit
/
```

CREATE_RESTRUCTURED_MODULE MEAPE Subcommand

Purpose Generates a restructuring procedure and executes the procedure to create a restructured load module on an object library. It can also save the restructuring procedure.

Format

CREATE_RESTRUCTURED_MODULE or
CRERM
RESTRUCTURED_MODULE=file
RESTRUCTURED_MODULE_NAME=any
RESTRUCTURING_COMMANDS=file
STATUS=status variable

Parameters **RESTRUCTURED_MODULE or RM**

File to which the object library containing the new load module is written. This parameter is required.

RESTRUCTURED_MODULE_NAME or RMN

Name given the new load module. If RESTRUCTURED_MODULE_NAME is omitted, the module name is the same as the file name.

RESTRUCTURING_COMMANDS or RC

File on which the restructuring procedure is saved. If RESTRUCTURING_COMMANDS is omitted, the restructuring procedure is discarded.

Remarks For more information, see the NOS/VE Object Code Management manual.

CREATE_RESTRUCTURING_COMMANDS MEAPE Subcommand

Purpose Generates and saves a restructuring procedure.

Format

CREATE_RESTRUCTURING_COMMANDS or CRERC
RESTRUCTURING_COMMANDS=file
RESTRUCTURED_MODULE=file
RESTRUCTURED_MODULE_NAME=any
STATUS=status variable

Parameters **RESTRUCTURING_COMMANDS or RC**

File to which the procedure is written. The procedure name is the same as the file name. This parameter is required.

RESTRUCTURED_MODULE or RM

Object library file to which the restructured module is written after the restructuring procedure is executed. This parameter is required.

RESTRUCTURED_MODULE_NAME or RMN

Name to be given the module created when the restructuring procedure is executed. If the RESTRUCTURED_MODULE_NAME parameter is omitted, the module name is the same as the file name.

Remarks

- The CREATE_RESTRUCTURING_COMMANDS subcommand uses the information accumulated in the connectivity matrix to generate the restructuring procedure.
- The CREATE_RESTRUCTURING_COMMANDS subcommand does not execute the restructuring procedure. To generate a restructured module, either enter a CREATE_RESTRUCTURED_MODULE subcommand during the session or execute the saved restructuring procedure.
- For more information, see the NOS/VE Object Code Management manual.

Examples

The following subcommand writes a restructuring procedure on file MODULE_RESTRUCTURE. If the procedure is executed, it creates a module named NEWLGO on object library file $USER.NEWLGO.

```
MPE/create_restructuring_commands restructuring_..
..MPE/commands=module_restructure restructured_..
..MPE/module=$user.newlgo
```

For an example of a restructuring procedure, refer to the BIND_MODULE subcommand description.

DISPLAY_PROGRAM_PROFILE MEAPE Subcommand

Purpose

Generates and displays a program profile. The program profile uses the execution time totals accumulated by previous EXECUTE_INSTRUMENTED_TASK subcommands.

Format

DISPLAY_PROGRAM_PROFILE or **DISPP**

PROFILE_ORDER=keyword
PROGRAM_UNIT_CLASS=keyword
NUMBER=integer or *keyword*
OUTPUT=file
STATUS=status variable

Parameters *PROFILE_ORDER* or *PO*

Order in which the program profile is displayed. Options are:

TIME (T)

By percentage of the total execution time ordered greatest to least.

PROGRAM_UNIT (PU)

By program unit name ordered alphabetically.

MODULE_PROGRAM_UNIT (MPU)

By module name ordered alphabetically.

If PROFILE_ORDER is omitted, TIME is used.

PROGRAM_UNIT_CLASS or *PUC*

Class of program units whose statistics are displayed. Options are:

ALL

All program units measured, both local and remote.

LOCAL

Only program units that are part of the target text.

REMOTE

Only program units that are called by target text program units, but are not part of the target text. These program units provide the remote block statistics in the program profile.

If PROGRAM_UNIT_CLASS is omitted, ALL is used.

NUMBER or *N*

Number of program unit statistics displayed. The statistics are sorted as specified by the PROFILE_ORDER parameter and then displayed in order until the specified number of statistics have been displayed. If NUMBER is omitted, the entire program profile is displayed.

OUTPUT or *O*

File to which the display is written. This file can be positioned. If OUTPUT is omitted, file $OUTPUT is used.

Remarks For more information, see the NOS/VE Object Code Management manual.

EXECUTE_INSTRUMENTED_TASK MEAPE Subcommand

Purpose Executes and measures the performance of the last program specified by a SET_PROGRAM_DESCRIPTION or SET_PROGRAM_MEASURES subcommand.

Format

EXECUTE_INSTRUMENTED_TASK or
EXEIT
PARAMETER=string
NO_CONNECTIVITY_MATRIX=boolean
WORKING_SET_INTERVAL=integer
STATUS=status variable

Parameters

PARAMETER or *P*

Parameter string passed to the program.

NO_CONNECTIVITY_MATRIX or *NCM*

Indicates whether a connectivity matrix is generated.

NOTE

Specify NO_CONNECTIVITY_MATRIX=TRUE if you do not intend to generate a restructuring procedure for the program. Omitting generation of a connectivity matrix saves time and system resources.

TRUE

No connectivity matrix is generated.

FALSE

A connectivity matrix is generated.

If NO_CONNECTIVITY_MATRIX is omitted, FALSE is assumed and a connectivity matrix is generated.

WORKING_SET_INTERVAL or *WSI*

Reserved.

Remarks

- The program is executed once for each EXECUTE_INSTRUMENTED_TASK subcommand you enter. You can specify a different parameter list for each execution. Cumulative statistics for all executions are kept.
- For more information, see the NOS/VE Object Code Management manual.

Examples The following sequence executes the modules on file LGO twice; cumulative statistics are kept for the program executions. The program profile is saved on file $USER.PROFILE_LIST.

```
/measure_program_execution
MPE/set_program_definition target_text=lgo
MPE/execute_instrumented_task parameter='size=40' ..
MPE../no_connectivity_matrix=true
MPE/execute_instrumented_task parameter='size=400' ..
MPE../no_connectivity_matrix=true
MPE/display_program_profile output=$user.profile_list
MPE/quit
/
```

QUIT
MEAPE Subcommand

Purpose Ends a MEASURE_PROGRAM_EXECUTION utility session.

Format **QUIT** or
QUI

Parameters None.

Remarks For more information, see the NOS/VE Object Code Management manual.

RESTORE_PROGRAM_MEASURES
MEAPE Subcommand

Purpose Restores the program measurement environment using the information saved by a SAVE_PROGRAM_MEASURES subcommand.

Format

RESTORE_PROGRAM_MEASURES or
RESPM
MEASURES=file
STATUS=status variable

Parameters

MEASURES or **M**

File containing a saved program measurement environment. This parameter is required.

Remarks

- The RESTORE_PROGRAM_MEASURES subcommand always restores the program description. It also restores the execution time statistics and connectivity matrix if that information was saved on the file.
- For more information, see the NOS/VE Object Code Management manual.

Examples

The following sequence begins a program measurement session and restores the program measurement environment saved on file SAVED_MEASUREMENT.

```
/measure_program_execution
MPE/restore_program_measures measures=..
MPE../saved_measurement
MPE/
```

SAVE_PROGRAM_MEASURES MEAPE Subcommand

Purpose

Saves the current program measurement environment on a file.

Format

SAVE_PROGRAM_MEASURES or
SAVPM
MEASURES=file
AMOUNT=list of keyword
STATUS=status variable

Parameters

MEASURES or M

File on which the program measurement environment is saved. This parameter is required.

AMOUNT or A

Information to be saved. Options are:

ALL

Program description, connectivity matrix, and execution time totals.

CONNECTIVITY_MATRIX (CM)

Program description and connectivity matrix only.

EXECUTION_TIME_TOTALS (ETT)

Program description and execution time totals only.

If AMOUNT is omitted, ALL is used.

Remarks

- By default, the SAVE_PROGRAM_MEASURES subcommand saves the execution time totals and the connectivity matrix. If the session that uses the saved program measurement environment will not use the execution time totals or connectivity matrix, you can direct the subcommand not to save that information with the AMOUNT parameter.
- The SAVE_PROGRAM_MEASURES subcommand does not discard the program measurement statistics. The statistics are discarded when you specify another program description or end the session.
- To use the saved program environment in another session, enter a RESTORE_PROGRAM_MEASURES subcommand that specifies the file containing the saved program environment.
- The program measures file is written as a sequential data file. It is not intended to be listed; its only intended use is to resume a MEASURE_PROGRAM_EXECUTION session.
- For more information, see the NOS/VE Object Code Management manual.

Examples The following subcommand copies the program description and any accumulated statistics to file SAVED_MEASUREMENT.

```
MPE/save_program_measure measures=saved_measurement
```

SET_PROGRAM_DESCRIPTION MEAPE Subcommand

Purpose Specifies the program whose performance is to be measured.

Format

SET_PROGRAM_DESCRIPTION or
SETPD
TARGET_TEXT=file
FILE=list of file
LIBRARY=list of file
MODULE=list of any
STARTING_PROCEDURE=any
STACK_SIZE=integer
STATUS=status variable

Parameters

TARGET_TEXT or **TT**

Object file or object library containing the modules to be measured. This parameter is required.

FILE or *FILES* or *F*

Object list for the program. Each module in the specified object files and object libraries is unconditionally included in the program. The list must include the target text file. If FILE is omitted, the object list for the program consists of only the file specified on the TARGET_TEXT parameter.

LIBRARY or *LIBRARIES* or *L*

List of object libraries added to the program library list.

MODULE or *MODULES* or *M*

Module list.

You use a string value for a module whose name is not an SCL name.

Each module is unconditionally loaded from the object libraries in the program library list.

STARTING_PROCEDURE or *SP*

Name of the entry point where execution begins.

You use a string value for an entry point whose name is not an SCL name.

If STARTING_PROCEDURE is omitted, the last transfer symbol encountered during loading is used.

STACK_SIZE or *SS*

Upper size limit in bytes of the run-time stack used for procedure call linkages and local variables. If STACK_SIZE is omitted, a 2-million byte limit is used.

Remarks

- The program to be measured should be debugged and ready for use in a production environment. The program description specified on the subcommand should be the same program description used to execute the program in a production environment.
- When you execute the SET_PROGRAM_DESCRIPTION subcommand, any program description previously in effect and any program measurement statistics accumulated for that program are discarded.
- For more information, see the NOS/VE Object Code Management manual.

Examples

The following subcommand specifies that program modules are on files LGO and SUBLGO but only the module on file LGO is to be measured.

```
MPE/set_program_description target_text=lgo ..
..MPE/file=(lgo,sublgo)
```

RECOVER_KEYED_FILE 18

RECOVER_KEYED_FILE Command

Purpose Begins a keyed-file recovery attempt.

Format

RECOVER_KEYED_FILE or
RECKF
FILE=file
PASSWORD=name
STATUS=status variable

Parameters

FILE or **F**

File path to the damaged keyed file to be recovered. This parameter is required.

If the damaged file does not currently exist, its cycle number cannot be determined by default. Therefore, the file path must explicitly specify the file cycle number so that the utility can reload the correct backup copy.

PASSWORD or *PW*

File password specified when Backup_Permanent_File wrote the backup copy of the file. A file password is optional, but, if a password exists for the file, it is required on this command. If no password exists for the file, NONE can be specified.

The file password in effect when the backup copy was written must be the same password in effect when the file was damaged. Otherwise, the backup copy cannot replace the damaged file.

Remarks

- The LOG_RESIDENCE attribute of the file specified on the command must match the LOG_RESIDENCE attribute of the backup copy to be reloaded. Recover_Keyed_File cannot use a backup copy that was written before the LOG_RESIDENCE attribute of the file was changed.

- If the file does not currently exist and the LOG_RESIDENCE of its backup copy is not the default log, you must enter a SET_FILE_ATTRIBUTE command for the file. The command must specify the same file cycle specified on the RECOVER_KEYED_FILE command and the same LOG_RESIDENCE as that of the backup copy to be used. (See the Example.)

- Similarly, if the file does not currently exist, but the file had a password when the backup copy was written, you must create the file with the same password. To do so, enter a CREATE_FILE command specifying the file path (including its cycle number) and the PASSWORD parameter.

- For more information, see the NOS/VE Advanced File Management Usage manual.

Examples

The following session attempts to restore a keyed file that no longer exists using its latest backup copy. When the latest backup copy was written, the file password was HUSH_HUSH and the LOG_RESIDENCE attribute was $USER.MY_LOG. Therefore, those values must be reestablished for the file cycle.

```
/recover_keyed_file, $user.keyed_file.1
reckf/create_file, $user.keyed_file.1, ..
reckf../password=hush_hush
reckf/set_file_attribute, $user.keyed_file.1, ..
reckf../log_residence=$user.my_log
reckf/recover_file_media
```

HELP
RECKF Subcommand

Purpose

Provides access to online information about the utility.

Format

HELP or
HEL
SUBJECT=string
MANUAL=file
STATUS=status variable

Parameters

SUBJECT or *S*

Topic to be found in the index of the online manual. The topic title must be enclosed in apostrophes ('topic').

If you omit the SUBJECT parameter, HELP displays a list of the available subcommands and prompts for display of a subcommand description in the online manual.

MANUAL or *M*

Online manual file to be read. If you omit the MANUAL parameter, the default is AFM. The working catalog is searched for the AFM file and then the $SYSTEM.MANUALS catalog.

Remarks

- If the SUBJECT parameter specifies a topic that is not in the manual index, a nonfatal error is returned notifying you that the topic could not be found.
- The default manual file, $SYSTEM.MANUALS.AFM, contains the online version of the NOS/VE Advanced File Management Usage manual, as provided with the NOS/VE system.
- If your terminal is defined for full-screen applications, the online manual is displayed in screen mode. Help is available for reading the online manual. To leave the online manual and return to the utility, use QUIT.
- For more information, see the NOS/VE Advanced File Management Usage manual.

Examples

The following session shows the default display returned by the HELP subcommand.

```
/recover_keyed_file, $user.keyed_file.1
reckf/help
The following Recover_Keyed_File subcommands are available:
RECOVER_FILE_MEDIA
VOID_LOG_FOR_RESTORED_FILE
HELP
QUIT

For a description of a subcommand in the online manual, enter:
HELP subject = '<subcommand>'

To return from an online manual, enter: QUIT
reckf/quit
/
```

QUIT
RECKF Subcommand

Purpose Ends the Recover_Keyed_File session.

Format

QUIT or
QUI
STATUS=status variable

Remarks

- The QUIT command is required to end a session.
- A recovery attempt that returns a fatal error ends the session.
- For more information, see the NOS/VE Advanced File Management Usage manual.

RECOVER_FILE_MEDIA
RECKF Subcommand

Purpose Reloads a backup of the file and then updates it using an update recovery log for the file.

Format

RECOVER_FILE_MEDIA or
RECFM
DAYS_SINCE_LAST_GOOD=integer
HOURS_SINCE_LAST_GOOD=integer
MINUTES_SINCE_LAST_GOOD=integer
STATUS=status variable

Parameters

DAYS_SINCE_LAST_GOOD or *DSLG*

Number of days since the damaged file was intact (any integer not less than 0). It is used with the next two parameters to determine the backup copy to be reloaded.

If the first three parameters are omitted, the default value for each is 0, causing the latest backup copy to be reloaded.

HOURS_SINCE_LAST_GOOD or *HSLG*

Number of hours (added to the days specified by the first parameter) since the damaged file was intact (an integer from 0 through 23).

If the first three parameters are omitted, the latest backup copy is reloaded.

MINUTES_SINCE_LAST_GOOD or *MSLG*

Number of minutes (added to the days and hours specified by the first two parameters) since the damaged file was intact (an integer from 0 through 59).

If the first three parameters are omitted, the latest backup copy is reloaded.

Remarks

- This subcommand is effective only if both a backup copy and an update recovery log are available for the file.
- An update recovery log is maintained for the file only if its LOGGING_OPTIONS attribute includes the option ENABLE_MEDIA_RECOVERY.
- The subcommand can only reload backup copies created by the Backup_Permanent_File utility because those backup copies are recorded in the update recovery log for the file. (The ENABLE_MEDIA_RECOVERY logging option must be set before the backup.)
- For a backup copy to be used, the file password (if any), the LOG_RESIDENCE attribute, and the LOGGING_OPTIONS attribute for the file must not have changed since the backup copy was written.
- The FILE_CLASS and INITIAL_VOLUME parameters are described in detail as parameters of the REQUEST_MASS_STORAGE command in the NOS/VE System Performance and Maintenance, Volume 2, Maintenance manual.
- Once a keyed file is recovered using RECOVER_FILE_MEDIA, it must be backed up (using the Backup_Permanent_File utility) before it can be updated.
- The subcommand issues progress messages as it proceeds. Be sure to read the messages as they appear.
- For more information, see the NOS/VE Advanced File Management Usage manual.

Examples The following session recovers the file using the last backup copy.

```
/recover_keyed_file, $user.my_keyed_file
reckf/recover_file_media
/
```

VOID_LOG_FOR_RESTORED_FILE RECKF Subcommand

Purpose Discards the update recovery log associated with a file that has been restored using the RESTORE_PERMANENT_FILE utility.

Format **VOID_LOG_FOR_RESTORED_FILE** or **VOILFRF**

STATUS=status variable

Remarks

- This subcommand is provided for situations in which an older version of the file is restored using the Restore_Permanent_File utility, and the user, content with this version, does not want to try to recover lost updates from the log.
- Updates cannot be recorded on a log associated with a restored file because the updates on the log do not correspond to the restored version of the file. (The restored file is an older version.) As a result, this subcommand is used to discard all past logged updates for the restored file.
- After the update recovery log is discarded, a backup copy of the file must be created by the Backup_Permanent_File utility if subsequent updates are to be recorded on the log.
- For more information, see the NOS/VE Advanced File Management Usage manual.

RESTORE_LOG 19

RESTORE_LOG Command

Purpose Begins a Restore_Log utility session.

Format

RESTORE_LOG or
RESL
LOG_RESIDENCE=file
STATUS=status variable

Parameters

LOG_RESIDENCE or **LR**

Catalog path containing the files composing the log to be restored. This parameter is required.

Remarks

- Immediately after entering the Restore_Log session, you should use the VALIDATE_LOG or RESTORE_REPOSITORIES subcommands to determine the type and extent of log damage, if any.
- For more information, see the NOS/VE Advanced File Management Usage manual.

DELETE_LOG_CONTROL_FILE RESL Subcommand

Purpose Deletes the log control file.

Format

DELETE_LOG_CONTROL_FILE or
DELLCF
STATUS=status variable

Remarks

- The log control file should be deleted only if it is damaged or if you want to force the log control file to be restored from the backup file. Damage to the log control file can be detected by the VALIDATE_LOG, RESTORE_REPOSITORIES, or RESTORE_LOG_CONTROL_FILE subcommands.
- For more information, see the NOS/VE Advanced File Management Usage manual.

DELETE_REPOSITORIES RESL Subcommand

Purpose Deletes log repositories.

Format

DELETE_REPOSITORIES or
DELETE_REPOSITORY or
DELR
REPOSITORIES=list of range of integer or keyword
STATUS=status variable

Parameters

REPOSITORIES or REPOSITORY or R

Specifies which repositories in the log are to be deleted. This parameter is required.

List of integer

Specifies the repositories to be deleted. Values can be a list of repository numbers specified in the repository name. Repositories have names in the format AAF$REPOSITORY_n where n is the integer value specified; that is, AAF$REPOSITORY_1, starting at one for the first repository, and incremented sequentially and contiguously. The last repository is specified as AAF$REPOSITORY_0. You can specify as many values as there are repositories to be deleted. If more than one value is specified, the values must be enclosed in parentheses and separated by commas or spaces.

ALL or A

All repositories in the log are deleted.

Remarks

- Repositories should be deleted only if they are damaged or if you want to force the repositories to be restored from the backup files. Damage to repositories can be detected by the VALIDATE_LOG or RESTORE_REPOSITORIES subcommands.
- For more information see the NOS/VE Advanced File Management Usage manual.

ENABLE_LOG
RESL Subcommand

Purpose Enables a disabled log; that is, makes the log available for general use.

Format

ENABLE_LOG or
ENAL
STATUS=status variable

Remarks

- If the log is disabled and it is usable; that is, the log is undamaged, ENABLE_LOG enables it. This makes the log available for general use.
- If the log is disabled but not usable, an error is displayed and the log remains disabled. Damage can be detected on the log control file and/or the repositories.
- A log must be enabled and usable before you can use it to recover keyed files.
- For more information see the NOS/VE Advanced File Management Usage manual.

HELP
RESL Subcommand

Purpose Provides access to online information about the utility.

Format

HELP or
HEL
SUBJECT=string
MANUAL=file
STATUS=status variable

Parameters

SUBJECT or *S*

Topic to be found in the index of the online manual. The topic must be enclosed in apostrophes ('topic').

If you omit the SUBJECT parameter, HELP displays a list of the available subcommands and prompts for display of a subcommand description in the online manual.

MANUAL or *M*

Online manual file to be read. If you omit the MANUAL parameter, the default is AFM. The subcommand searches for the file in the working catalog and then in the $SYSTEM.MANUALS catalog.

Remarks

- If the SUBJECT parameter specifies a topic that is not in the manual index, a nonfatal error is returned notifying you that the topic could not be found.
- The default manual file, $SYSTEM.MANUALS.AFM, contains the online version of the NOS/VE Advanced File Management Usage manual, as provided with the NOS/VE system.
- If your terminal is defined for full-screen applications, online manuals are displayed in screen mode. Help on reading online manuals is available in the online manual. To leave the online manual and return to the utility, use QUIT.
- For more information see the NOS/VE Advanced File Management Usage manual.

Examples The following session shows the default display returned by the HELP subcommand.

```
/restore_log
resl/help

The following Restore_Log subcommands are available:

VALIDATE_LOG
RESTORE_REPOSITORIES
RESTORE_LOG_CONTROL_FILE
DELETE_REPOSITORIES
DELETE_LOG_CONTROL_FILE
ENABLE_LOG
HELP
QUIT

For the description of a subcommand in the online
manual, enter:  HELP subject = '<subcommand>'

To return from an online manual, enter:  QUIT
resl/quit
/
```

QUIT
RESL Subcommand

Purpose Ends the Restore_Log session.

Format

QUIT or
QUI
STATUS=status variable

Remarks

- The QUIT command is required to end a session.
- For more information see the NOS/VE Advanced File Management Usage manual.

RESTORE_LOG_CONTROL_FILE RESL Subcommand

Purpose Restores the log control file from the specified log backup file.

Format

RESTORE_LOG_CONTROL_FILE or
RESLCF
MEDIA=keyword
BACKUP_FILE=file
EXTERNAL_VSN=list of string
RECORDED_VSN=list of string
TYPE=keyword
STATUS=status variable

Parameters

MEDIA or **M**

Device class of the log backup file to be restored. This parameter is required.

MAGNETIC_TAPE_DEVICE or MTD

Indicates that the log backup file is stored on a labeled tape. (In this case, the BACKUP_FILE parameter is not used.)

MASS_STORAGE_DEVICE or MSD

Indicates that the log backup file specified by the BACKUP_FILE parameter is stored on disk. (In this case, the RECORDED_VSN, EXTERNAL_VSN, and TYPE parameters are not used.)

BACKUP_FILE or *BF*

The file path name of one of the backup files in the log (previously established by the CONFIGURE_LOG_BACKUP subcommand of the Administer_Recovery_Log utility) to be used for restoring the log control file. This parameter must be specified if MEDIA is set to MASS_STORAGE_DEVICE.

EXTERNAL_VSN or *EVSN*

List of external VSNs identifying the tape volumes that compose the log backup file. The VSNs are specified as strings of from 1 through 6 characters enclosed in apostrophes.

RECORDED_VSN or *RVSN*

List of recorded VSNs of the tape volumes that compose the log backup file. The recorded VSN is in the ANSI VOL1 label on the volume. The VSNs are specified as strings of from 1 through 6 characters enclosed in apostrophes. This parameter must be specified if MEDIA is set to MAGNETIC_TAPE_DEVICE.

TYPE or *T*

Tape density of the nine-track tape drive on which the log backup file was written.

MT9$800

Indicates 800 cpi.

MT9$1600

Indicates 1600 cpi.

MT9$6250

Indicates 6250 cpi.

The default value is MT9$6250.

Remarks

- In general, the backup file that was written to most recently is the best one to specify first as the log backup file. If RESTORE_LOG_CONTROL_FILE fails, try again specifying the next most recent backup file, and so on.
- The log control file can be restored only if the log was configured for log backups (see the CONFIGURE_LOG_BACKUP subcommand of the Administer_Recovery_Log utility). A copy of the log control file exists at the front of each log backup file, having been written there as part of the ongoing process of backing up the log.
- If the log control file is not already disabled, RESTORE_LOG_CONTROL_FILE immediately disables it. This is to ensure the log is not used while it is being restored. The log can be enabled using ENABLE_LOG (described later in this chapter).

- RESTORE_LOG_CONTROL_FILE restores a log control file only if it detects damage to the log control file. Damage to the log control file can also be detected by the RESTORE_REPOSITORIES or VALIDATE_LOG subcommands.

- Once a damaged log control file is restored, the log is no longer available for logging entries. The log is available only for recovering keyed files. To begin logging entries again, you must switch to a different log, or you must delete the log whose log control file has been restored, then recreate it.

- For more information see the NOS/VE Advanced File Management Usage manual.

RESTORE_REPOSITORIES RESL Subcommand

Purpose Restores damaged repository log files from the log backup files.

Format

RESTORE_REPOSITORIES or
RESR
STATUS=status variable

Remarks

- Older repositories can be restored only if the log was configured for automatic backups (see CONFIGURE_LOG_BACKUPS of the Administer_Recovery_Log utility). If the active repository is to be replaced, backups are not required.

- If the log is not already disabled, RESTORE_REPOSITORIES immediately disables it. This is to ensure that the log is not used while it is being restored. Once the log is restored, it can be enabled using ENABLE_LOG.

- Initially, RESTORE_REPOSITORIES determines the usability of the log; that is, the type and extent of log damage, if any.

- Once the log is restored, if recovery information is lost (for example, the active repository is lost, which had not yet been backed up), or if the log control file has been restored, the log is available only for recovery operations. To begin recording log entries again, you must switch to a different log, or you must delete the log, then recreate it.
- For more information see the NOS/VE Advanced File Management Usage manual.

VALIDATE_LOG RESL Subcommand

Purpose Determines the usability of the log; that is, the type and extent of log damage, if any.

Format

VALIDATE_LOG or
VALL
STATUS=status variable

Remarks

- If damage to the log is detected and if the log is not already disabled, VALIDATE_LOG immediately disables it. This is to ensure that the log is not used while it is being restored. Once the log is restored, it can be enabled using ENABLE_LOG. If no damage to the log is detected, the log is not disabled.
- For more information see the NOS/VE Advanced File Management Usage manual.

RESTORE_PERMANENT_FILES 20

RESTORE_PERMANENT_FILES Command

Purpose Initiates the utility that restores permanent files and catalogs from backup copies created by the BACKUP_PERMANENT_FILE utility. The restore operations are directed by RESTORE_PERMANENT_FILE subcommands.

Format

RESTORE_PERMANENT_FILES or
RESTORE_PERMANENT_FILE or
RESPF
LIST=file
STATUS=status variable

Parameters *LIST* or *L*

Identifies the file to which a summary of the results of the restore utility are written and, optionally, specifies how the file is to be positioned prior to use. Omission causes $LIST to be used.

Remarks

- The content of the list file can be specified using the SET_LIST_OPTION subcommand prior to using a RESTORE_PERMANENT_FILE subcommand. If the SET_LIST_OPTION subcommand is omitted, the modification date and time and size of the file are displayed for each permanent file cycle.
- For more information, see the NOS/VE System Usage manual.

Examples The following subcommand initiates a RESTORE_PERMANENT_FILE subcommand utility session. The subcommand specifies that the report listing be written to file RESTORE_LISTING.

```
/restore_permanent_files list=restore_listing
```

Following entry of this subcommand, RESTORE_PERMANENT_FILE subcommands can be entered in response to the following prompt.

```
PUR/
```

$BACKUP_FILE
RESPF Function

Purpose Returns a string containing information on a backup file produced by the BACKUP_PERMANENT_FILE utility. Because this function causes the file to be rewound, only the first item of information found on the file can be queried and returned to you. When the string value is returned, all letters within the string are converted to uppercase. This function is valid only within the RESTORE_PERMANENT_FILE utility.

Format

$BACKUP_FILE or
$BF
(file
keyword)

Parameters

file

Specifies the name of the backup file to be queried. This parameter is required.

keyword

Specifies the particular attribute that is being queried. The following are valid keywords.

IDENTIFIER (I)

Returns a string containing the path name of the first name on the backup file.

IDENTIFIER_TYPE (IT)

Returns a string containing a name that indicates the type of the first item on the backup file. One of the following names is returned.

SET, CATALOG, FILE, CYCLE

If you do not specify a keyword, IDENTIFIER is assumed.

Remarks

- This function is especially useful when attempting to restore from a backup file for which the destination is known but the name of the file or catalog is unknown.
- The $BACKUP_FILE function always returns a string. The $FNAME function is included in the RESTORE_CATALOG command to convert this string to a file name. Once the string has been converted to a file name, you can use the file name in any subsequent RESTORE_FILE or RESTORE_CATALOG subcommands.
- For more information, see the NOS/VE System Usage manual.

Examples

For the following example, assume that you receive a backup tape produced by the BACKUP_CATALOG command and you wish to restore the catalog to your own $USER.MY_CATALOG. To do this, enter the following commands.

```
/restore_permanent_files l=list_file
PUR/restore_catalog $fname($backup_file(backup_file,..
PUR../identifier)) backup_file=backup_file ..
PUR../new_catalog_name=$user.my_catalog
PUR/quit
```

DISPLAY_BACKUP_FILE RESPF Subcommand

Purpose

Displays the contents of a backup file.

Format

DISPLAY_BACKUP_FILE or
DISBF
BACKUP_FILE=file
DISPLAY_OPTION=keyword
NUMBER=integer or *keyword*
STATUS=status variable

Parameters **BACKUP_FILE or BF**

Specifies the file that contains the backup copies of the files and catalogs previously backed up by a BACKUP_PERMANENT_FILE utility session.

DISPLAY_OPTION or DO

Specifies the level of information to be displayed. Options are:

IDENTIFIER (I)

Displays the name and type (file or catalog) of each entry on the backup file.

DESCRIPTOR (D)

Displays the following information:

- Record headers maintained on the backup file.
- Version of the backup utility that produced the backup file.
- Date and time the backup file was written.
- Backup utility subcommand that produced the backup file.
- Cycle number of each file cycle.
- Usage count of each file cycle.
- Creation date and time of each file cycle.
- Last access date and time of each file cycle.
- Date and time of the last modification of each file cycle.
- Expiration date of each file cycle.
- Size of each file cycle.

READ_DATA (RD)

Displays the information described for the DESCRIPTOR parameter and also attempts to read all data for each cycle on the backup file. The listing

reports whether or not the data is read without error. No attempt is made to verify the data with the original file backed up.

If omitted, IDENTIFIER is assumed.

NUMBER or *N*

Selects the number of catalogs, files, or cycles from the beginning of the backup file for which information is to be displayed. If this parameter is omitted or if the keyword value ALL is specified, all entries on the backup file are displayed.

Remarks For more information, see the NOS/VE System Usage manual.

QUIT RESPF Subcommand

Purpose Ends a RESTORE_PERMANENT_FILES utility session.

Format

QUIT or
QUI

Parameters None.

Remarks For more information, see the NOS/VE System Usage manual.

RESTORE_ALL_FILES RESPF Subcommand

Purpose Enables a system operator to restore all catalogs and all permanent files for a NOS/VE system (those written to the backup file with the BACKUP_ALL_FILES subcommand). Other users can restore all catalogs which they own and all files and cycles for which they have cycle permission.

Format

RESTORE_ALL_FILES or
RESAF
BACKUP_FILE=file
STATUS=status variable

Parameters

BACKUP_FILE or BF

Specifies the file that contains the backup copies of the files and catalogs to be restored. This parameter is required.

Remarks

- Backup copies of catalogs and files that do not already exist in the permanent file system are restored.
- Catalogs and files that already exist are not altered.
- The file specified by the BACKUP_FILE parameter is initially positioned to beginning-of-information.
- To restore permanent files when partial backups have been taken, the RESTORE_ALL_FILES subcommand is used to restore the last partial backup first. This has the effect of restoring the catalog structure as it was at the time of the last partial backup. File cycle data that is not contained on the last partial back is restored using the RESTORE_EXCLUDED_FILE_CYCLES subcommand.
- For more information, see the NOS/VE System Usage manual.

Examples

The following job restores all files in the system that were previously backed up with a BACKUP_ALL_FILES subcommand.

```
/job
job/request_magnetic_tape file=pf_tape_file ..
job../evsn='pfb001' type=mt9$6250
job/restore_permanent_files
job/restore_all_files backup_file=pf_tape_file
job/quit
job/jobend
```

RESTORE_CATALOG RESPF Subcommand

Purpose Restores a catalog that does not currently exist as a catalog.

Format

RESTORE_CATALOG or
RESC
CATALOG=file
BACKUP_FILE=file
NEW_CATALOG_NAME=file
STATUS=status variable

Parameters

CATALOG or **C**

Specifies the catalog that is to be restored from the backup file. This parameter is required.

BACKUP_FILE or **BF**

Specifies the file that contains the backup copy of the catalog and its associated files and subcatalogs. This parameter is required.

NEW_CATALOG_NAME or *NCN*

Specifies the catalog into which the files and subcatalogs on the backup file are restored. Omission causes the name as it exists on the backup file to be used.

Remarks

- Backup copies of files and subcatalogs are restored.
- You must be the owner of the catalog.
- This command cannot be used to restore your master catalog.
- The catalog being restored must not currently exist.
- The file specified by the BACKUP_FILE parameter must have been created by the BACKUP_PERMANENT_FILE utility.
- The backup file is initially positioned at beginning-of-information.
- For more information, see the NOS/VE System Usage manual.

Examples The following example restores the master catalog to a new subcatalog in the master catalog.

```
/restore_permanent_files list=restore_listing
PUR/restore_catalog catalog=$user new_catalog_name=..
PUR../$user.catalog_2 backup_file=backed_up_files
PUR/quit
```

RESTORE_EXCLUDED_FILE_CYCLES RESPF Subcommand

Purpose Restores cycles to files that currently exist in the permanent file system but do not have data defined for them.

Format

RESTORE_EXCLUDED_FILE_CYCLES or
RESTORE_EXCLUDED_FILE_CYCLE or
RESEFC
FILE=file
CATALOG=file
BACKUP_FILE=file
NEW_NAME=file
RESTORE_OPTIONS=list of keyword
STATUS=status variable

Parameters

FILE or *F*

The FILE parameter specifies the file or cycle for which data is to be restored (as identified on the backup file). If no cycle number is specified, data for all cycles of the file is restored. If specified, a cycle number must be a specific cycle (not $HIGH or $LOW).

CATALOG or *C*

The CATALOG parameter specifies the catalog for which data is to be restored (as identified on the backup file). Data for all cycles in the catalog is restored.

BACKUP_FILE or **BF**

Specifies the file containing the backup information. This file is positioned at the beginning-of-information. This parameter is required.

NEW_NAME or *NN* or *NEW_CATALOG_NAME* or *NCN* or *NEW_FILE_NAME* or *NFN*

Specifies a new name for the catalog, file, or cycle for which the data is being restored. This parameter can be used if the name on the backup file is different than that in the current permanent file system. Omission causes the name as it exists on the backup file to be used. If a cycle reference was included on the FILE parameter but not on the NEW_NAME parameter, $HIGH is used.

RESTORE_OPTIONS or *RESTORE_OPTION* or *RO*

Reserved for site personnel.

Remarks

- This subcommand is used to restore cycle data when partial backups have been performed. If the permanent file system is backed up by a full backup followed by daily partial backups, then the last partial backup is restored with the RESTORE_ALL_FILES subcommand. All other backups are restored in reverse order using this subcommand.
- The modification date on the backup file must match the modification date in the current permanent file catalog, unless otherwise specified by a SET_RESTORE_OPTIONS subcommand.
- If a cycle already has data defined for it, the cycle is not altered.
- You may specify either the file or catalog parameter, but not both. Omission of both parameters causes all data to be restored, in which case the NEW_NAME parameter cannot be used.
- For more information, see the NOS/VE System Usage manual.

Examples

The following example restores files from a previous partial dump and a previous full dump.

```
/restore_permanent_files
PUR/restore_all_files bf=partial_dump
PUR/restore_excluded_file_cycles bf=full_dump
```

The following example restores all cycles of a file from a partial and full dump.

```
PUR/restore_file $user.data_file_1 bf=partial_dump
PUR/resefc file=$user.data_file_1 bf=full_dump
```

RESTORE_EXISTING_CATALOG RESPF Subcommand

Purpose Restores the contents of a currently existing catalog.

Format

RESTORE_EXISTING_CATALOG or
RESEC
CATALOG=file
BACKUP_FILE=file
NEW_CATALOG_NAME=file
STATUS=status variable

Parameters

CATALOG or **C**

Specifies the catalog that is to be restored from the backup file. This parameter is required.

BACKUP_FILE or **BF**

Specifies the file that contains the backup copy of the catalog and its associated files and subcatalogs. The file is initially positioned at beginning-of-information. This parameter is required.

NEW_CATALOG_NAME or *NCN*

Specifies the existing catalog into which the files and subcatalogs on the backup file are restored. Omission causes the name as it exists on the backup file to be used.

Remarks

- Backup copies of files and subcatalogs that do not already exist in the specified catalog are restored.
- Any cycle that already exists is not altered.
- Cycle permission is required to restore any file cycle within an existing catalog.
- You must be the owner of the catalog to restore any subcatalogs.
- The file specified by the BACKUP_FILE parameter must have been created by the BACKUP_PERMANENT_FILE utility.

- The backup file is initially positioned at beginning-of-information.
- For more information, see the NOS/VE System Usage manual.

Examples The following commands restore the master catalog that was backed up with the BACKUP_CATALOG subcommand.

```
/restore_permanent_files list=restore_list
PUR/restore_existing_catalog ..
PUR../catalog=$user backup_file=backed_up_files
PUR/quit
```

RESTORE_EXISTING_FILE RESPF Subcommand

Purpose Restores the file cycles of an existing file.

Format

RESTORE_EXISTING_FILE or
RESEF
FILE=file
BACKUP_FILE=file
PASSWORD=name or *keyword*
NEW_FILE_NAME=file
STATUS=status variable

Parameters

FILE or F

Specifies the file whose file cycles are to be restored from the backup file. If a cycle reference is included the cycle reference is ignored. This parameter is required.

BACKUP_FILE or BF

Specifies the file that contains the backup copy of the file. This parameter is required.

PASSWORD or *PW*

Specifies the file password. This parameter must match the password of the existing file. Omission or specifying the keyword NONE causes no password to be used.

NEW_FILE_NAME or *NFN*

Specifies the existing file to be restored. Omission causes the name as it exists on the backup file to be used.

Remarks

- All file cycles that exist on the backup file but do not exist as a permanent file are restored.
- Cycles that currently exist as permanent files are not altered.
- You must have CYCLE permission to restore an existing file.
- The file specified by the BACKUP_FILE parameter must have been created during by BACKUP_PERMANENT_FILE utility.
- The backup file is initially positioned at beginning-of-information.
- For more information, see the NOS/VE System Usage manual.

Examples The following example restores cycle number 87 of file DATA_FILE_0 in subcatalog CATALOG_1 of the master catalog that was previously backed up.

```
/delete_file $user.catalog_1.data_file_0.87 ..
../pw=new_data_0_pw
/respf
PUR/restore_existing_file ..
PUR../$user.catalog_1.data_file_0 ..
PUR../bf=copy_of_file pw=new_data_0_pw
PUR/quit
```

RESTORE_FILE RESPF Subcommand

Purpose Restores the file cycles of a file that does not currently exist as a permanent file.

Format

RESTORE_FILE or
RESF
FILE=file
BACKUP_FILE=file
PASSWORD=name or *keyword*
NEW_FILE_NAME=file
STATUS=status variable

Parameters

FILE or **F**

Specifies the file whose file cycles are to be restored from the backup file. This parameter is required.

BACKUP_FILE or **BF**

Specifies the file that contains the backup copy of the file. This parameter is required.

PASSWORD or *PW*

Specifies the file password. It must match the existing file password. This parameter is used only if a specific cycle of an existing file is being restored. Omission or specifying the keyword NONE causes no password to be used.

NEW_FILE_NAME or *NFN*

Specifies a new name for the file being restored. Omission causes the name as it exists on the backup file to be used.

Remarks

- If the file name includes a cycle reference, only that cycle is restored (at least one file cycle must already exist).
- If a cycle reference is omitted, all file cycles are restored (the file must not already exist).
- If a cycle reference is included on the FILE parameter, it must be a specific cycle number; the keywords $HIGH and $LOW cannot be used.
- If a cycle reference is not specified on the NEW_FILE_NAME parameter, $NEXT is used.
- If a cycle reference is specified on the NEW_FILE_NAME parameter, the file specified with the FILE parameter must also include a cycle reference.
- You must have CYCLE permission to the file in order to restore all file cycles or an additional file cycle.
- The file specified by the BACKUP_FILE parameter must have been created by the BACKUP_PERMANENT_FILE utility.

- The backup file is initially positioned at beginning-of-information.
- For more information, see the NOS/VE System Usage manual.

Examples The following subcommands restore cycle number 87 of file DATA_FILE_0 in subcatalog CATALOG_1. The file is restored as cycle number 1 of file DATA_FILE_2 in CATALOG_2 of the master catalog.

```
/respf
PUR/restore_file $user.catalog_1.data_file_0.87 ..
PUR../bf=copy_of_file pw=new_data_0_pw ..
PUR../nfn=$user.catalog_2.data_file_2
PUR/quit
```

SET_LIST_OPTIONS RESPF Subcommand

Purpose Specifies the information that is written to the list file by subsequent subcommands.

Format

SET_LIST_OPTIONS or
SET_LIST_OPTION or
SETLO
FILE_DISPLAY_OPTIONS=list of keyword
CYCLE_DISPLAY_OPTIONS=list of keyword
DISPLAY_EXCLUDED_ITEMS=boolean
STATUS=status variable

Parameters *FILE_DISPLAY_OPTIONS* or *FILE_DISPLAY_OPTION* or *FDO*

Selects the data to be displayed with the file name. Options are:

ACCOUNT (A)

Displays the account name.

PROJECT (P)

Displays the project name.

NONE

Displays only the file name.

ALL

Displays the account and project name.

If the FILE_DISPLAY_OPTION parameter is omitted, NONE is selected.

CYCLE_DISPLAY_OPTIONS or *CYCLE_DISPLAY_OPTION* or *CDO*

Selects the data to be displayed for each cycle backed up or restored. The cycle number and whether the cycle was excluded is also displayed. Options are:

CREATION_DATE_TIME (CDT)

Displays the date and time the cycle was created.

ACCESS_DATE_TIME (ADT)

Displays the date and time the cycle was last accessed.

MODIFICATION_DATE_TIME (MDT)

Displays the date and time the cycle was last modified.

EXPIRATION_DATE (ED)

Displays the expiration date of the cycle.

ACCESS_COUNT (AC)

Displays the number of accesses to the cycle.

SIZE (S)

Displays the size of the cycle in bytes.

RECORDED_VSN (RVSN)

Displays the disk volumes on which the cycle resides.

GLOBAL_FILE_NAME (GFN)

Displays the internally generated global file name. This name is neither backed up nor restored.

NONE

Displays the only cycle number.

ALL

Selects all of the display options.

If the CYCLE_DISPLAY_OPTION parameter is omitted, the MODIFICATION_DATE_TIME and SIZE options are used.

DISPLAY_EXCLUDED_ITEMS or *DISPLAY_EXCLUDED_ITEM* or *DEI*

Determines whether or not excluded catalogs, files, and cycles are displayed on the list file. TRUE causes the identification of all excluded catalogs, files, and cycles to be displayed. If FALSE is specified, excluded items are not displayed. Omission causes TRUE to be used.

Remarks For more information, see the NOS/VE System Usage manual.

SOURCE_CODE_UTILITY 21

SOURCE_CODE_UTILITY Command

Purpose Begins an SCU command utility session.

Format

SOURCE_CODE_UTILITY or
SCU or
SOUCU
STATUS=status variable

Remarks

- Entering a CREATE_LIBRARY or USE_LIBRARY subcommand initializes the working library for the SCU command utility session. If neither subcommand is issued, file SOURCE_LIBRARY is used for the base and result libraries. If file SOURCE_LIBRARY does not exist, it is created.
- For more information, see the NOS/VE Source Code Management manual.

Examples The following sequence begins an SCU session and initializes the working library from file OLDPL in your working catalog, assumed not to be $LOCAL. The base file, OLDPL, is a source file whose file structure is a library. Entering the QUIT subcommand causes the working library to be written on the next cycle of file OLDPL.

```
/source_code_utility
sc/use_library base=oldpl result=oldpl.$next
sc/quit
```

The next example does not use the USE_LIBRARY subcommand, but rather initializes the working library from file SOURCE_LIBRARY in your working catalog.

```
/source_code_utility
sc/create_deck deck=deck1 ..
sc../modification=version1
sc/quit
```

ADD_LIBRARY
SCU Subcommand

Purpose Adds decks from one or more source libraries to the working library.

Format

ADD_LIBRARY or
ADD_LIBRARIES or
ADDL
SOURCE_LIBRARY=list of file
LIST=file
DISPLAY_OPTIONS=keyword
STATUS=status variable

Parameters

SOURCE_LIBRARY or **SOURCE_LIBRARIES** or **SL**

List of one or more source library files. This parameter is required.

LIST or *L*

Listing file. You can specify a file position as part of the file name. SCU lists the source library origin of each deck in the working library. Within an SCU session, if you omit LIST, the listing file is the file specified on the SET_LIST_OPTIONS subcommand. Otherwise, the default is file $LIST.

DISPLAY_OPTIONS or *DO*

Specifies the level of information listed. Currently, both keyword values produce the same listing.

BRIEF or B
FULL or F

If DISPLAY_OPTIONS is omitted, BRIEF is used.

Remarks

- ADD_LIBRARIES only adds decks that are not already in the working library. It reads the deck list for each source library in the order you specify the libraries on the command. When it reads a deck name that is not currently in the working library, it adds the deck to the library. When it reads a deck name that is already in the working library, it sends a message describing the duplication, but it does not add the deck to the working library.

- If a modification is in more than one source library modification list and the creation times do not match, ADD_LIBRARY reports an error and does not add any decks to the working library.
- If no decks could be merged because an exception occurred in each deck, an error status is returned and ADD_LIBRARY makes no change to the library.
- Decks, features, groups, and modifications are ordered alphabetically on the ADD_LIBRARIES result library.
- Key characters in source libraries that are added to the working library must match the key character in the working library. If the key characters do not match, SCU generates an error message.
- For more information, see the NOS/VE Source Code Management manual.

Examples The following command adds the decks from the source library on file $USER.NEWLIB to the working library. The contents of the working library are then displayed.

```
sc/add_library $user.newlib list=output

DECKA                          BASE
DECKB                          BASE
DECKC                          NEWLIB
DECKD                          BASE
```

CHANGE_DECK SCU Subcommand

Purpose Changes the content of one or more deck header fields.

Format

CHANGE_DECK or
CHANGE_DECKS or
CHAD
DECK=list of name or *keyword*
AUTHOR=string
CLEAR_ORIGINAL_INTERLOCK=boolean
CLEAR_SUB_INTERLOCK=boolean
DECK_DESCRIPTION=list of string
PROCESSOR=string
GROUP=list of name

DELETE_GROUP=list of name
CHARACTER=string or *keyword*
TAB_COLUMN=list of integer
DELETE_COLUMN=list of integer
WIDTH=integer
LINE_IDENTIFIER=keyword
EXPAND=boolean
STATUS=status variable

Parameters *DECK* or *DECKS* or *D*

Decks whose headers are changed. You can specify a list of one or more names, a list of one or more ranges, or the keyword ALL. ALL specifies all decks in the library. The default is the name of the most recently used deck.

AUTHOR or *A*

New author. If AUTHOR is omitted, the author field is not changed.

CLEAR_ORIGINAL_INTERLOCK or *COI*

Indicates whether the original interlock for an extracted deck should be cleared. Options are:

TRUE

Clears the original interlock field of the extracted deck by erasing the name and time stamp that were recorded in this deck.

FALSE

Leaves the original interlock field of the extracted deck unchanged.

If CLEAR_ORIGINAL_INTERLOCK is omitted, FALSE is used.

CLEAR_SUB_INTERLOCK or *CLEAR_INTERLOCK* or *CI* or *CSI*

Indicates whether the subinterlock field of the original deck should be cleared. Options are:

TRUE

Clears the subinterlock field of the original deck.

FALSE

Leaves the subinterlock field of the original deck unchanged.

If CLEAR_SUB_INTERLOCK or CLEAR_INTERLOCK is omitted, FALSE is used.

NOTE

You must have authority 4 for the file to clear a deck subinterlock or original interlock field.

DECK_DESCRIPTION or *DD*

List of strings containing the new deck description. If DECK_DESCRIPTION is omitted, the description field is not changed.

PROCESSOR or *P*

New processor. If PROCESSOR is omitted, the processor field is not changed.

GROUP or *GROUPS* or *G*

Additional groups to which the deck is to belong. The subcommand deletes any groups specified on the DELETE_GROUP parameter before adding groups to the group list. If GROUP is omitted, the deck is not associated with additional groups.

DELETE_GROUP or *DELETE_GROUPS* or *DG*

Groups to which the deck should no longer belong. The subcommand deletes groups specified before adding any groups specified on the GROUP parameter. If DELETE_GROUP is omitted, the deck continues to belong to the same groups it did previously.

CHARACTER or *C*

Either a 1-character string containing the new default tab character or the keyword NONE to disable tabbing. If CHARACTER is omitted, the tabbing status and default tab character are not changed.

TAB_COLUMN or *TAB_COLUMNS* or *TC*

List of from 1 to 256 additional default tab columns. SCU deletes the tab columns on the DELETE_COLUMN parameter before it adds the new tab columns. If TAB_COLUMN is omitted, no new tab columns are added.

DELETE_COLUMN or *DELETE_COLUMNS* or *DC*

List of default tab columns or tab column ranges to be removed. SCU deletes the specified tab columns before it adds the tab columns on the TAB_COLUMN parameter. If DELETE_COLUMN is omitted, no tab columns are removed.

WIDTH or *W*

New default line width. If WIDTH is omitted, the default line width is not changed.

LINE_IDENTIFIER or *LI*

New default line identifier placement. Options are:

RIGHT (R)

Place line identifiers to the right of the text.

LEFT (L)

Place line identifiers to the left of the text.

NONE

No line identifiers are placed on output lines.

If LINE_IDENTIFIER is omitted, the default line identifier placement is not changed.

EXPAND or *E*

New expand attribute value. Options are:

TRUE

An EXPAND_DECK subcommand expands the deck. (The deck can also be expanded by a COPY or COPYC directive.)

FALSE

An EXPAND_DECK subcommand does not expand the deck; it skips the deck and continues processing at the next deck. Only a COPY or COPYC directive can expand the deck.

If EXPAND is omitted, the expand attribute is not changed.

Remarks

- The DECK parameter specifies each deck to which the changes should apply. The other parameters (except STATUS) specify the deck header fields to be changed.
- To display a deck header, enter a DISPLAY_DECK subcommand. You can reference deck header fields with the SCU function $DECK_HEADER.
- If you have access authority 4 for the file, you can enter a CHANGE_DECK subcommand to clear a subinterlock that was set when a user extracted a deck from the library.
- To eliminate unused groups from a library, enter EXTRACT_SOURCE_LIBRARY DECKS=ALL INTERLOCK=NONE to copy all decks to a new RESULT file, saving only groups, modifications, and features belonging to those decks.
- Changes to a deck header are not part of any modification. When you include or exclude modifications, you must make any associated deck header changes separately by entering the CHANGE_DECK subcommand.
- For more information, see the NOS/VE Source Code Management manual.

Examples

The following subcommand adds default tab column 35 and deletes default tab column 30 for DECK1.

```
sc/change_deck deck=deck1 tab_column=35 delete_column=30
```

The following subcommand clears the subinterlock fields of all deck headers in the working library if you have access authority 4 for the file.

```
sc/change_deck all clear_interlock=true
```

CHANGE_DECK_NAME SCU Subcommand

Purpose Substitutes new names for existing deck names.

Format

CHANGE_DECK_NAME or
CHANGE_DECK_NAMES or
CHADN
NAME_LIST=file
LIST=file
CHANGE_DECK_REFERENCES=boolean
MODIFICATION=name
STATUS=status variable

Parameters

NAME_LIST or **NL**

Name substitution file. This parameter is required.

LIST or *L*

Listing file. You can specify a file position as part of the file name. If LIST is omitted, the listing file is the file specified on the SET_LIST_OPTIONS subcommand. Otherwise, the default is file $LIST.

CHANGE_DECK_REFERENCES or *CDR*

Indicates whether the command substitutes deck names on COPY and COPYC directives. Options are:

TRUE

COPY and COPYC names are substituted.

FALSE

COPY and COPYC names are not substituted.

If CHANGE_DECK_REFERENCES is omitted, FALSE is used.

MODIFICATION or *M*

Modification to which the changed lines belong. If MODIFICATION is omitted, SCU$ALTER is used.

Remarks

- A deck name can occur in two places within a source library: within its deck header, and on COPY and COPYC directives in the source text. To list the COPY and COPYC references to the deck, enter a DISPLAY_DECK_REFERENCES command.

- You store the name substitutions on a separate file and specify the file on the NAME_LIST parameter. Each name substitution is specified as a line containing an SCL parameter list. The parameter list must have the following parameters:

  OLD_NAME (ON)

  Existing name.

  NEW_NAME (NN)

  Substituted name. NEW_NAME must be different from ALL.

- For more information, see the NOS/VE Source Code Management manual.

Examples The following subcommand changes deck names as specified in file NEW_DECK_NAMES. The changed lines belong to the default modification SCU$ALTER.

```
sc/change_deck_names name_list=new_deck_names ..
sc../change_deck_references=true
```

The contents of file NEW_DECK_NAMES are:

```
my_deck,deck465
```

The command replaces each occurrence of the deck name MY_DECK with the new name DECK465. Because the command specifies that the CHANGE_DECK_REFERENCES parameter is TRUE, it replaces the deck name both in the deck header and on COPY and COPYC directives throughout the library.

CHANGE_DECK_REFERENCES SCU Subcommand

Purpose Changes the deck names of COPY and COPYC directives that are located in the specified decks.

Format

CHANGE_DECK_REFERENCES or
CHADR
DECK=list of name
MODIFICATION=name
NAME_LIST=file
LIST=file
STATUS=status variable

Parameters

DECK or *DECKS* or *D*

Decks in which substitutions are performed. The keyword ALL specifies all decks in the library. If DECK is omitted, ALL is used.

MODIFICATION or *M*

Modification to which the changed lines belong. If MODIFICATION is omitted, SCU$ALTER is used.

NAME_LIST or **NL**

Name substitution file. This parameter is required.

LIST or *L*

Listing file. You can specify a file position as part of the file name. If LIST is omitted, the listing file is the file specified on the SET_LIST_OPTIONS subcommand. Otherwise, the default is file $LIST.

Remarks

- The CHANGE_DECK_REFERENCES subcommand only changes deck names on COPY and COPYC directives, not in deck headers. To change a deck name in its deck header, enter the CHANGE_DECK_NAMES command.
- You use CHANGE_DECK_REFERENCES to replace references to one deck with references to another deck. To list the COPY and COPYC references to a deck, enter a DISPLAY_DECK_REFERENCES command.
- You store the name substitutions on a separate file and specify the file on the NAME_LIST parameter. Each name substitution is specified as a line containing an SCL parameter list. The parameter list must have the following parameters:

  OLD_NAME (ON)

  Existing name.

  NEW_NAME (NN)

  Substituted name. NEW_NAME should be different from ALL.

- For more information, see the NOS/VE Source Code Management manual.

Examples The following subcommand changes references as specified in file NEW_NAMES. The changes belong to modification RENAME.

```
sc/change_deck_references name_list=new_names ..
sc../modification=rename
```

The following lists the contents of file NEW_NAMES.

```
deck44,deck45
```

The command changes each COPY or COPYC reference to DECK44 so that it references DECK45.

CHANGE_LIBRARY SCU Subcommand

Purpose Changes the content of one or more fields in the working library header.

Format

CHANGE_LIBRARY or
CHAL
LIBRARY=name
LIBRARY_DESCRIPTION=list of string
VERSION=string
LAST_USED_DECK=name
LAST_USED_MODIFICATION=name
STATUS=status variable

Parameters *LIBRARY* or *L*

New library name. If LIBRARY is omitted, the library name is not changed.

LIBRARY_DESCRIPTION or *LD*

Strings used to describe the source code that is maintained on this library. If LIBRARY_DESCRIPTION is omitted, the description field is not changed.

VERSION or *V*

New library version. If VERSION is omitted, the version field is not changed.

LAST_USED_DECK or *LUD*

Default deck name that is stored in the library header. The deck name is used as the default value for the deck parameter on most subcommands. Specifying NONE clears

the last used deck name. If a name is explicitly stated for a DECK parameter on an SCU subcommand, LAST_USED_DECK is automatically changed.

LAST_USED_MODIFICATION or *LUM*

Default modification name that is stored in the library header. The modification name is used as the default value for the modification parameter on most subcommands. Specifying NONE clears the last used modification name. If a name is explicitly stated for a MODIFICATION parameter on an SCU subcommand, LAST_USED_MODIFICATION is automatically changed to that name.

Remarks

- To display the contents of the library header, enter a DISPLAY_LIBRARY command.
- You can reference library header fields with the SCU function $LIBRARY_HEADER.
- For more information, see the NOS/VE Source Code Management manual.

Examples The following command changes the content of the library version field.

```
sc/change_library version='Version 1.1'
```

CHANGE_MODIFICATION SCU Subcommand

Purpose Changes information in one or more modification descriptions.

Format

CHANGE_MODIFICATION or
CHANGE_MODIFICATIONS or
CHAM
MODIFICATION=list of name or *keyword*
FEATURE=name or *keyword*
AUTHOR=string
MODIFICATION_DESCRIPTION=list of string
STATE=integer
STATUS=status variable

Parameters

MODIFICATION or *MODIFICATIONS* or *M*

Modification descriptions to be changed. You can specify a list of one or more names (from 1 to 9 characters each), a list of one or more ranges, or the keyword ALL. ALL specifies all modifications in the library. If MODIFICATION is omitted, the information for the description of the last used modification is changed.

FEATURE or *F*

New feature name or keyword NONE. Specifying NONE clears the current feature association. If FEATURE is omitted, the feature field is not changed.

AUTHOR or *A*

New author. If AUTHOR is omitted, the author field is not changed.

MODIFICATION_DESCRIPTION or *MD*

Strings used to describe the modifications. If MODIFICATION_DESCRIPTION is omitted, the description field is not changed.

STATE or *S*

New modification state. The following are the states and their descriptions.

State	Description
0	Experimental
1	Developmental
2	Stable
3	Verified
4	Released

If STATE is omitted, the state is not changed.

NOTE

You cannot raise the modification state above your authority for the file.

Remarks

- The CHANGE_MODIFICATIONS subcommand can only change the headers of modifications within the modification list of the working library.
- To raise the value in the state field of the modification header, your authority for the library file must be the same or greater than the new state. For example, to raise the state to 2, your authority must be 2, 3, or 4.

  You can only lower a state to 0. To lower the state to 0, your authority for the library file must be the same or greater than the current state. For example, to lower a modification that is currently in state 2, your authority must be 2, 3, or 4.
- To display a modification header, enter a DISPLAY_MODIFICATION command. You can reference modification header fields with the SCU function $MODIFICATION_HEADER.
- To eliminate unused groups from a library, enter EXTRACT_SOURCE_LIBRARY DECKS=ALL INTERLOCK=NONE to copy all decks to a new RESULT file, saving only groups, modifications, and features that belong to these decks.
- The FEATURE name should not be a keyword.
- For more information, see the NOS/VE Source Code Management manual.

Examples

The following command clears the feature associations of all modifications in the working library.

```
sc/change_modification all feature=none
```

The following command raises the state of MOD_4 to state 1 (developmental). You must have at least authority 1 for the file to raise the modification state to 1.

```
sc/change_modification mod_4 state=1
```

COMBINE_LIBRARY SCU Subcommand

Purpose Combines the decks from one or more source libraries with those in the working library.

Format

COMBINE_LIBRARY or
COMBINE_LIBRARIES or
COML
SOURCE_LIBRARY=list of file
LIST=file
DISPLAY_OPTIONS=keyword
ENFORCE_INTERLOCKS=boolean
STATUS=status variable

Parameters

SOURCE_LIBRARY or **SOURCE_LIBRARIES** or **SL**

List of one or more source library names. This parameter is required.

LIST or *L*

Listing file. You can specify a file position as part of the file name. SCU lists the source library origin of each deck in the working library. If LIST is omitted, the listing file is the file specified on the SET_LIST_OPTIONS subcommand. Otherwise, the default is file $LIST.

DISPLAY_OPTIONS or *DO*

Specifies the information listed. Currently, both of the following keywords produce the same listing.

BRIEF or B
FULL or F

If DISPLAY_OPTIONS is omitted, BRIEF is used.

ENFORCE_INTERLOCKS or *EI*

Indicates whether the original interlock field of a source library deck must match the subinterlock field of the working library deck it is to replace. Options are:

TRUE

Interlocks must match.

FALSE

Interlocks need not match.

If ENFORCE_INTERLOCKS is omitted, FALSE is used.

Remarks

- COMBINE_LIBRARY reads the source library deck lists in the order you specify the libraries on the command.
- After reading a deck name, COMBINE_LIBRARY determines if the deck name is already in the working library deck list. If the name is not in the list, it adds the deck to the working library. If the name is already in the list, it replaces the deck in the working library with the deck from the source library. The combining process is continued until each successive source library in the list has been combined with the working library.
- If no decks could be merged because an exception occurred in each deck, an error status is returned and no change is made to the library.

  If the creation times of modifications that occur on both libraries do not match, COMBINE_LIBRARY issues an error and does not alter the working library.
- COMBINE_LIBRARY lists the source library origin of each deck in the working library on the listing file.
- Decks, features, groups, and modifications are ordered alphabetically on the COMBINE_LIBRARY result library.
- You can enter a COMBINE_LIBRARY subcommand to merge decks from an extracted library with the decks in the library from which it was extracted to form a new library. It adds new decks and replaces existing decks.
- If you set interlocks when you extract the library, entering COMBINE_LIBRARY enforces the interlock if you specify that the interlocks should be enforced. COMBINE_LIBRARY checks whether the original interlock value in the extracted deck header matches the subinterlock value in the working library header. If the values match, the working library deck is

replaced with the extracted deck. Otherwise, it issues a warning messsage, does not replace the working library, and then attempts to combine any remaining decks in the list.

- Key characters in source libraries that are added to the working library must match the key character in the working library. If the key characters do not match, SCU generates an error message.
- For more information, see the NOS/VE Source Code Management manual.

Examples The following subcommand combines the decks in the source library NEWLIB with the decks in the working library.

```
sc/combine_library newlib list=output

DECKA                          BASE
DECKB                          BASE
DECKC                          NEWLIB
DECKD                          BASE
DECKE                          NEWLIB
```

CREATE_DECK SCU Subcommand

Purpose Creates one or more decks.

Format

CREATE_DECK or
CREATE_DECKS or
CRED
DECK=list of name
MODIFICATION=name
SOURCE=list of file
AUTHOR=string
DECK_DESCRIPTION=list of string
PROCESSOR=string
GROUP=list of name
CHARACTER=string or *keyword*
TAB_COLUMN=list of integer
WIDTH=integer
LINE_IDENTIFIER=keyword

EXPAND=boolean
DECK_DIRECTIVES_INCLUDED=boolean
MULTI_PARTITION=boolean
SAME_AS=name
STATUS=status variable

Parameters

DECK or *DECKS* or *D*

List of one or more deck names. Each name must be unique to the library. If DECK is omitted, you must specify the SOURCE parameter and DECK_DIRECTIVES_INCLUDED=TRUE.

MODIFICATION or *M*

Modification name (1 to 9 characters). The modification must be in state 0 (zero). The default is the last used modification.

SOURCE or *SOURCES* or *S*

List of one or more files containing the source text for the decks. You can specify a file position as part of the file name. The SOURCE parameter is required when you specify DECK_DIRECTIVES_INCLUDED=TRUE.

AUTHOR or *A*

Optional author identification.

DECK_DESCRIPTION or *DD*

List of strings containing the optional deck description. If DECK_DESCRIPTION is omitted, a description is not saved.

PROCESSOR or *P*

Optional identification of the processor to which the deck text is input.

GROUP or *GROUPS* or *G*

Optional list of groups to which the deck is to belong. If any of the group names are not in the group list, SCU adds the names to the list.

CHARACTER or *C*

Either a 1-character string containing the tab character or the keyword NONE to disable tabbing. If CHARACTER is omitted, tabbing is disabled.

TAB_COLUMN or *TAB_COLUMNS* or *TC*

Optional list of 1 through 256 default tab columns. The column numbers range from 1 through 256.

WIDTH or *W*

Default line width. If WIDTH is omitted or specified as 0 (zero), deck lines can be up to 256 characters and the lines are not padded with trailing blanks when the deck is expanded.

LINE_IDENTIFIER or *LI*

Default line identifier placement.

RIGHT (R)

Identifiers are placed to the right of the text.

LEFT (L)

Identifiers are placed to the left of the text.

NONE

No line identifiers are placed on output lines.

If LINE_IDENTIFIER is omitted, NONE is used.

EXPAND or *E*

Specifies the expand attribute for the decks created. Applicable only if the subcommand names decks on its DECK parameter, not if DECK directives name the decks. (A DECK directive specifies the expand attribute for its deck.)

TRUE

An EXPAND_DECK subcommand expands the deck. (COPY and COPYC directives can also expand the deck.)

FALSE

An EXPAND_DECK subcommand skips the deck and continues its processing with the next specified deck. Only COPY and COPYC directives can expand the deck.

If EXPAND is omitted, TRUE is used.

DECK_DIRECTIVES_INCLUDED or *DDI*

Indicates whether the deck names are specified on DECK directives embedded in the source text or as the DECK parameter of this subcommand.

TRUE

The deck names are on DECK directives in the source text on the source file. CREATE_DECK only reads text from the first source file specified when DECK directives are included.

FALSE

The deck names shown in the DECK parameter.

If DECK_DIRECTIVES_INCLUDED is omitted, FALSE is used and the DECK parameter must be specified.

MULTI_PARTITION or *MP*

Indicates whether the deck text can be more than one partition of data.

TRUE

The subcommand can copy more than one partition of data to each deck.

FALSE

The subcommand can copy only one partition of data to each deck.

If MULTI_PARTITION is omitted, FALSE is used.

SAME_AS or *SA*

Optional deck name. If a name is specified, the subcommand copies deck header fields not specified on the CREATE_DECK subcommand from the deck header of this deck. If SAME_AS is omitted, unspecified header fields are left blank.

Remarks

- CREATE_DECK provides a header for each deck. The minimum content of the deck header is the deck name and the creation modification. You can specify additional values for deck header fields with parameters on the subcommand. You can also specify the SAME_AS parameter to copy deck header fields from another deck header; CREATE_DECK only copies those deck header fields not explicitly specified.

- Each deck created is given a name (from 1 through 31 characters). By default, the subcommand uses the deck names specified on the DECK parameter. However, if you specify DECK_DIRECTIVES_INCLUDED=TRUE on the subcommand, it uses the deck names specified on DECK directives in the source text. You can specify the EXPAND attribute for a deck on its DECK directive.

- The subcommand can specify the creation modification for the deck. A modification name is from 1 through 9 characters, and it can be an existing modification within the library or a new modification. Any source text that the subcommand copies to a deck belongs to the creation modification. The default is the last used modification.

- To copy source text to the newly created decks, you must specify the SOURCE parameter. If you specify the SOURCE parameter and the DECK parameter, you must specify a file name for each deck name on the DECK parameter. The subcommand copies text to each deck from its corresponding file on the SOURCE parameter; that is, it copies the text from the first file to the first deck created, the text from the second file to the second deck created, and so forth. If you specify the file $NULL for a deck, the subcommand copies no text and the deck remains empty.

- By default, the subcommand copies only the first partition of text from a source text file. To copy more than one partition of text, specify MULTI_PARTITION=TRUE on the subcommand. This indicates that if the subcommand reads an end-of-partition delimiter when copying text, it converts the delimiter to a WEOP text-embedded directive and continues copying text.

- If you specify DECK_DIRECTIVES_INCLUDED=TRUE and omit the DECK parameter, the subcommand creates a deck header for each DECK directive it reads on the source text file.
- If you specify DECK_DIRECTIVES_INCLUDED=TRUE and errors are encountered in the source file, CREATE_DECK attempts to skip ahead to the next DECK directive. The working library will contain the decks that were processed without errors.
- The subcommand places the created decks within the library so that the alphabetic sequence of names in the deck list is maintained.
- The maximum number of lines in one deck is 16,777,214.
- For more information, see the NOS/VE Source Code Management manual.

Examples

The following subcommand creates two decks. First, it creates a deck named DECK2 and copies one partition of text to the deck from file FILE2. It then creates a deck named DECK3 and copies one partition of text to the deck from file FILE3. The deck headers contain the same information as the DECK1 header, except for their description fields.

```
sc/create_deck (deck2,deck3) modification=original ..
sc../source=(file2,file3) same_as=deck1 ..
sc../deck_description='Second version of INIT_ARRAY'
```

The following subcommand creates decks using the text on file FILE4. SCU generates a deck header for each DECK directive embedded in the file text. The deck headers are the same as the DECK1 header, except for the name and expand attribute fields. The DECK directive specifies the deck name and expand attribute.

```
sc/create_deck modification=original source=file4 ..
sc../same_as=deck1 deck_directives_included=true
```

CREATE_LIBRARY SCU Subcommand

Purpose Creates an empty source library at the beginning of an SCU session. This subcommand also specifies the result library used during an SCU session.

Format

CREATE_LIBRARY or
CREL
RESULT=file
LIBRARY=name
LIBRARY_DESCRIPTION=list of string
KEY=string
VERSION=string
STATUS=status variable

Parameters

RESULT or *R*

Name of the file to be used as the result file during an SCU session. If RESULT is omitted, the file SOURCE_LIBRARY in your working catalog is used as the result file.

LIBRARY or *L*

Library name. If LIBRARY is omitted, the name specified by the RESULT parameter is used as the library name.

LIBRARY_DESCRIPTION or *LD*

String or strings that describe the source code maintained on this library. If LIBRARY_DESCRIPTION is omitted, the null string is used.

KEY or *K*

One-character string containing the key character. The key character is the first character of a text-embedded directive. If KEY is omitted, * is used.

VERSION or *V*

String used to describe the version of the library. If VERSION is omitted, the null string is used.

Remarks

- Using the CREATE_LIBRARY subcommand, you can specify a key character other than the default character *. The key character is the character SCU recognizes as the prefix for all text-embedded directives in the library.

- CREATE_LIBRARY creates a source library containing only a library header, which you can display with the DISPLAY_LIBRARY subcommand. To change library header information, enter a CHANGE_LIBRARY subcommand. To reference a library header field, use the SCU function $LIBRARY_HEADER.

- After you execute CREATE_LIBRARY, the base library is selected and cannot be changed by a subsequent USE_LIBRARY or CREATE_LIBRARY subcommand.

- During an SCU session, if neither a CREATE_LIBRARY nor a USE_LIBRARY subcommand is issued before other subcommands, file SOURCE_LIBRARY in your working catalog is used for the base and result libraries.

- For more information, see the NOS/VE Source Code Management manual.

Examples The following sequence creates an empty source library named SOURCE_LIBRARY. The key character for the library is *.

```
/scu
sc/create_library
sc/quit
```

CREATE_MODIFICATION SCU Subcommand

Purpose Creates one or more modifications in the library modification list.

Format

CREATE_MODIFICATION or
CREATE_MODIFICATIONS or
CREM
MODIFICATION=list of name
FEATURE=name
AUTHOR=string
MODIFICATION_DESCRIPTION=list of string
STATUS=status variable

Parameters

MODIFICATION or **MODIFICATIONS** or **M**

List of one or more modification names (from 1 through 9 characters each). This parameter is required.

FEATURE or *F*

Optional name of the feature to which the modification belongs. If the feature name is not in the feature list, SCU adds the name to the list.

AUTHOR or *A*

Optional modification author.

MODIFICATION_DESCRIPTION or *MD*

Optional list of strings containing the modification description.

Remarks

- A modification created by a CREATE_MODIFICATION subcommand contains only the modification header; no lines belong to the modification. The modification is defined for specification on subsequent commands.
- Modifications are placed on the library in alphabetical order.
- If CREATE_MODIFICATIONS creates more than one header, the headers are identical except for their names.
- To display the modifications defined within the working library, enter a DISPLAY_MODIFICATION_LIST command. To determine within an expression whether a modification exists, use the SCU function $MODIFICATION.
- FEATURE name should not be ALL or NONE.
- For more information, see the NOS/VE Source Code Management manual.

Examples

The following subcommand creates a description for modification MOD_4 for feature SYNTAX_CHECK. The author of the modification is K. Riley. The text in the SCL variables LINE1 and LINE2 is the modification description.

```
sc/line1='This is a very long title for ..
sc../a modification to show that'
sc/line2='a list of strings may be used for ..
sc../the description.'
sc/create_modification modification=mod_4 ..
sc../feature=syntax_check author='K. Riley' ..
sc../modification_description=(line1,line2)
```

$BASE
SCU Function

Purpose Returns the base library file.

Format **$BASE**

Parameters None.

Remarks For more information, see the NOS/VE Source Code Management manual.

Examples The following command displays the current value of the base file.

```
/scu
sc/use_library base=$user.fortran_lib
sc/display_value value=$base
$USER.FORTRAN_LIB
```

$DECK
SCU Function

Purpose Returns a boolean value indicating whether the specified deck is in the working library.

Format **$DECK**
(name)

Parameters **name**

Name of the deck to be found. This parameter is required.

Remarks For more information, see the NOS/VE Source Code Management manual.

Examples The following command assigns a boolean value to the SCL variable DECK_EXISTS, depending on whether DECK1 is in the working library.

```
sc/deck_exists = $deck(deck1)
```

$DECK_HEADER
SCU Function

Purpose Returns the contents of any deck header field.

Format

$DECK_HEADER
(name
keyword)

Parameters

name

Name of the deck whose header field is returned. This parameter is required.

keyword

Deck header field. This parameter is required. Options are:

AUTHOR (A)

Deck author. The value is returned as a string.

ORIGINAL_INTERLOCK (OI)

Original interlock on the deck. The value is returned as a string.

SUB_INTERLOCK (SI)

Deck subinterlock. The value is returned as a string.

DECK_DESCRIPTION (DD)

Deck description. The value is returned as a string.

PROCESSOR (P)

Deck processor. The value is returned as a string.

GROUP or GROUPS (G)

Groups to which the deck belongs. The value is returned as an array of strings if the number of groups is greater than 1; otherwise the value returned is a single string.

CHARACTER (C)

Default tab character. The value is returned as a string.

TAB_COLUMN or TAB_COLUMNS (TC)

Default tab stop columns. The value is returned as a string.

WIDTH (W)

Default line width. The value is returned as an integer.

LINE_IDENTIFIER (LI)

Default line identifier placement. This value is returned as a string.

EXPAND (E)

Expand attribute. The value is returned as a boolean value.

MODIFICATION (M)

Names of the modifications that apply to the deck. The value is returned as an array of strings.

CREATION_DATE (CD)

Date the deck was created. The value is returned as a string (mm/dd/yy).

CREATION_TIME (CT)

Time the deck was created. The value is returned as a string (hh.mm.ss).

MODIFICATION_DATE (MD)

Date the deck was last changed. The value is returned as a string (mm/dd/yy).

MODIFICATION_TIME (MT)

Time the deck was last changed. The value is returned as a string (hh.mm.ss).

ACTIVE_LINE_COUNT (ALC)

Number of active lines in the deck. The value is returned as an integer.

INACTIVE_LINE_COUNT (ILC)

Number of inactive lines in the deck. The value is returned as an integer.

Remarks For more information, see the NOS/VE Source Code Management manual.

Examples The following command assigns the active line count for deck DECK5 to the SCL integer variable LINE_COUNT.

```
sc/line_count = $deck_header(deck5,alc)
```

$DECK_LIST
SCU Function

Purpose Returns an array of strings listing the names of decks on a library.

Format **$DECK_LIST**

Parameters None.

Remarks

- The names in the array will be ordered alphabetically as the decks are ordered on the library.
- When used in the selection criteria subcommand processing, $DECK_LIST reflects the current deck list to be written to the compile, result, or source file being produced.
- For more information, see the NOS/VE Source Code Management manual.

Examples This example shows an array implicitly created and values assigned to it. The first command assigns an array of strings to the variable DECK_LIST and the following commands execute a loop to display the values of the array.

```
sc/deck_list = $deck_list
sc/for i=1 to $variable(deck_list,upper_bound) do
for/display_value deck_list(i)
for/forend
DECK1
DECK2
DECK3
DECK4
sc/
```

DELETE_DECK SCU Subcommand

Purpose Deletes one or more decks from the working library.

Format

DELETE_DECK or
DELETE_DECKS or
DELD
DECK=list of range of name
STATUS=status variable

Parameters

DECK or DECKS or D

Decks to be deleted. This parameter is required.

Remarks

- You cannot delete a deck if the creation modification of the deck is in a state greater than your authority for the file.
- The DELETE_DECK subcommand removes the deck name from the deck list of the working library (as opposed to being deactivated like the EDIT_DECK DELETE_LINE subcommand).
- When you specify a range of decks, DELETE_DECK deletes each deck in the deck list, beginning with the first deck specified through the last deck specified. Before specifying a range of decks to be deleted, you should display the deck list with a DISPLAY_DECK_LIST subcommand to determine the decks included in the range.
- If a deck to be deleted has a conflicting subinterlock set, SCU sends a warning message, observing that another user extracted the deck using an EXTRACT_SOURCE_LIBRARY command. The deck is deleted. SCU then attempts to delete any remaining decks.

- For more information, see the NOS/VE Source Code Management manual.

Examples The following command deletes deck DECKA and decks DECKC through DECKF.

```
sc/delete_decks (decka,deckc..deckf)
```

DELETE_MODIFICATION SCU Subcommand

Purpose Deletes one or more modifications. Deleting a modification reverses all text changes that were introduced by the modification. All insertions are deleted, all replacements are removed, and all deletions are reactivated.

Format

DELETE_MODIFICATION or
DELETE_MODIFICATIONS or
DELM
MODIFICATION=list of range of name
DECK=list of range of name
STATUS=status variable

Parameters

MODIFICATION or **MODIFICATIONS** or **M**

Modifications to be deleted. This parameter is required.

DECK or *DECKS* or *D*

Either one or more deck names or the keyword ALL. ALL specifies all decks in the working library. If DECK is specified, SCU deletes only the modification changes within the specified decks. If DECK is omitted, ALL is used.

Remarks

- You cannot delete the creation modification of a deck directly: you must first delete each deck for which the modification is the creation modification. You can then delete the modification from the modification list.
- You cannot delete a modification whose state is greater than your authority for the file.

- If a deck affected by a deleted modification has its subinterlock set, SCU sends a warning message, stating that a user has extracted the deck with an EXTRACT_SOURCE_LIBRARY command. The modification is deleted. SCU then attempts deletion of modification changes on any remaining decks in the deck list.

- You can use this subcommand to create a new library without the modification. To temporarily reverse modification changes when expanding text, use the selection criteria subcommand EXCLUDE_MODIFICATION.

- For more information, see the NOS/VE Source Code Management manual.

Examples The following subcommand deletes modification MOD5.

```
sc/delete_modification mod5
```

DISPLAY_DECK
SCU Subcommand

Purpose Displays one or more deck headers.

Format

DISPLAY_DECK or
DISPLAY_DECKS or
DISD
DECK=list of name or *keyword*
OUTPUT=file
DISPLAY_OPTIONS=keyword
TEXT=keyword
STATUS=status variable

Parameters *DECK* or *DECKS* or *D*

Decks whose headers are to be displayed. You can specify a list of one or more deck names, a list of one or more deck ranges, or the keyword ALL. ALL specifies all decks in the working library. If DECK is omitted, the last used deck is displayed.

OUTPUT or *O*

File on which the display is written. You can specify a file position as part of the file name. If OUTPUT is omitted, file $OUTPUT is used.

DISPLAY_OPTIONS or *DO*

Specifies the information listed. Options are:

BRIEF (B)

Lists only deck header information.

FULL (F)

Lists deck header information, modifications to which deck lines belong, and the groups to which the deck belongs.

If DISPLAY_OPTIONS is omitted, BRIEF is used.

TEXT or *T*

Specifies deck text to be displayed. Options are:

INACTIVE (I)

Active and inactive lines.

ACTIVE (A)

Active lines only.

NONE

Deck text is not displayed.

If TEXT is omitted, NONE is used.

Remarks

- You can display deck text with the DISPLAY_DECK subcommand. You can display either the active lines or both the active and inactive lines. Inactive lines are lines that have been deleted; only active lines appear in expanded deck text.
- The DISPLAY_DECK subcommand is valid within an editing session started by an EDIT_DECK subcommand. It is also valid within a selection criteria file if prefixed with the slant character (/DISPLAY_DECK).
- For more information, see the NOS/VE Source Code Management manual.

Examples The following subcommand displays the deck header of deck DECK1. The subcommand specifies full information level (DO=F) so the modifications in the deck and the groups to which the deck belong are also displayed. The subcommand also specifies a listing of both the inactive and active lines in the deck (T=I).

```
sc/display_deck deck=deck1 display_options=f text=i
Deck Information

DECK: DECK1
EXPAND: FALSE
AUTHOR: M.J.Perreten
PROCESSOR: Fortran
ORIGINAL_INTERLOCK:
SUB_INTERLOCK:
WIDTH: 80
LINE IDENTIFIER: none
TAB ACTIVE: TRUE
CHARACTER: #
TAB_COLUMNS:   5,   7,   9,  11,  13  15,  17
CREATION_DATE - TIME: 12/02/81 - 10:41:51
MODIFICATION_DATE -TIME: 03/24/82 - 13:37:19
DECK_DESCRIPTION: First example deck
COUNTS
 MODS:2        GROUPS:1      LINES ACTIVE:6      INACTIVE:1

 MODS AND SEQUENCE NUMBERS
ORIGINAL  6
FIRST_MOD 1

 GROUP LIST
LOOPS
  Active(A)/Inactive(I) text lines for deck DECK1

A ORIGINAL        1
A ORIGINAL        2         DO 10 I=1,100
I ORIGINAL        3         10 I= I+1
                     I FIRST_MOD
A FIRST_MOD       1      10 I= I+1
A ORIGINAL        4
A ORIGINAL        5 *COPYC COMMON1
A ORIGINAL        6
```

Each line of the text listing contains a letter indicating whether the line is active or inactive (A or I), the line identifier, and the line text. If the line is inactive, the succeeding line names the modification that deactivated the line.

DISPLAY_DECK_LIST SCU Subcommand

Purpose Lists the decks found in the working library in alphabetical order by deck name.

Format

DISPLAY_DECK_LIST or
DISDL
ALTERNATE_BASE=list of file
OUTPUT=file
DISPLAY_OPTIONS=keyword
STATUS=status variable

Parameters

ALTERNATE_BASE or *ALTERNATE_BASES* or *AB*

Optional list of one or more source libraries whose deck lists are combined with the working library deck list. If ALTERNATE_BASE is omitted, the decks on the current working library will be displayed.

OUTPUT or *O*

File on which the display is written. You can specify a file position as part of the file name. If OUTPUT is omitted, file $OUTPUT is used.

DISPLAY_OPTIONS or *DO*

Specifies the information listed. Currently, both of the following keywords produce the same listing.

BRIEF (B)
FULL (F)

If DISPLAY_OPTIONS is omitted, BRIEF is used.

Remarks

- If you specify one or more alternate base libraries, DISPLAY_DECK_LIST combines their deck lists with the working library deck list for the duration of the subcommand. You can use this option to display the deck list that would be used if you specified the alternate base libraries on an EXPAND_DECKS or EXTRACT_DECKS subcommand.
- For more information, see the NOS/VE Source Code Management manual.

Examples The following subcommand displays a combined deck list of the decks on source library MY_LIB and the working library.

```
sc/display_deck_list alternate_base=my_lib
FORTRAN_TEXT      FORTRAN_TEXT_II
MY_TEXT
```

The listing does not indicate which source library contains the deck.

DISPLAY_DECK_REFERENCES SCU Subcommand

Purpose Displays a cross-reference listing for one or more decks. A reference to a deck is a COPY or COPYC directive that names the deck.

Format

DISPLAY_DECK_REFERENCES or
DISDR

DECK=list of name or *keyword*
EXTERNAL_DECK=list of name or *keyword*
OUTPUT=file
DECK_RESIDENCE=keyword
REFERENCE_DIRECTION=keyword
REFERENCE_TYPE=keyword
STATUS=status variable

Parameters *DECK* or *DECKS* or *D*

Decks to be cross-referenced. You can specify a list of names, a list of ranges, or the keyword ALL or NONE. ALL specifies all decks in the working library. If DECK is omitted, the name of the last deck is used. If you specify NONE, you prevent the last deck from being cross-referenced.

EXTERNAL_DECK or *EXTERNAL_DECKS* or *ED*

Decks to be cross-referenced that are not on the working library. You can specify a list of names or the keyword ALL. ALL specifies all decks not in the working library that are referenced by decks in the working library. If EXTERNAL_DECK is omitted, you must specify the DECK parameter.

OUTPUT or *O*

File on which the cross-reference is written. You can specify a file position as part of the file name. If OUTPUT is omitted, file $OUTPUT is used.

DECK_RESIDENCE or *DR*

Specifies the references to list. Options are:

EXTERNAL

List only references to decks not in the working library.

INTERNAL

List only references to decks in the working library.

ALL

List references to decks both in the working library and not in the working library.

If DECK_RESIDENCE is omitted, ALL is used.

REFERENCE_DIRECTION or *RD*

Specifies the direction the references are traced. Options are:

TO

References to the decks.

FROM

References from the decks.

ALL

References to and from the decks.

If REFERENCE_DIRECTION is omitted, TO is used.

REFERENCE_TYPE or *RT*

Specifies the reference type to be listed. Options are:

DIRECT

Lists only direct references.

INDIRECT

Lists only indirect references.

ALL

Lists both direct and indirect references.

If REFERENCE_TYPE is omitted, ALL is used.

Remarks

- The REFERENCE_TYPE parameter indicates whether DISPLAY_DECK_REFERENCES lists direct references or indirect references or both.
- Direct references involve only two decks; indirect references involve three or more decks. For example, if DECKA contains a COPY directive that copies DECKB, DECKA directly references DECKB. If DECKB contains a COPY directive that copies DECKC, DECKA indirectly references DECKC.
- The DECK_RESIDENCE parameter indicates whether this subcommand lists references to decks within the working library, decks not in the working library, or both.
- This subcommand is valid within an editing session started by an EDIT_DECK subcommand. It is also valid within a selection criteria file if prefixed with the slant character (/DISPLAY_DECK_ REFERENCES).
- For more information, see the NOS/VE Source Code Management manual.

Examples The following subcommand produces a cross-reference for deck SUB1 on the working library. It traces direct and indirect references both to and from the deck, including references to decks not resident on the working library.

```
sc/display_deck_references deck=sub1 ..
sc../reference_direction=all
  References FROM deck
  (e = external deck, i = indirect reference)

SUB1                                  references
 e  SUB2

  References TO internal deck
  (i = indirect reference)

SUB1                                  is referenced by
   PROGRAM1
```

DISPLAY_FEATURE SCU Subcommand

Purpose Displays the modifications belonging to a feature.

Format

DISPLAY_FEATURE or
DISF
FEATURE=name
OUTPUT=file
DISPLAY_OPTIONS=keyword
STATUS=status variable

Parameters

FEATURE or **F**

Feature name. This parameter is required.

OUTPUT or *O*

File on which the display is written. You can specify a file position as part of the file name. If OUTPUT is omitted, file $OUTPUT is used.

DISPLAY_OPTIONS or *DO*

Specifies the information displayed. Options are:

BRIEF (B)

Lists only the modification names.

FULL (F)

Lists the modification names and the modification descriptions.

If DISPLAY_OPTIONS is omitted, BRIEF is used.

Remarks

- You can change the feature to which a modification belongs with the CHANGE_MODIFICATION subcommand.
- The DISPLAY_FEATURE subcommand is valid within an editing session started by an EDIT_DECK subcommand. It is also valid within a selection criteria file if prefixed with the slant character (/DISPLAY_FEATURE).
- For more information, see the NOS/VE Source Code Management manual.

Examples

The following subcommand displays the names and modification descriptions for all modifications belonging to the feature NEW_PROMPTS.

```
sc/display_feature new_prompts display_options=f
Descriptions of modifications associated with the feature
   NEW_PROMPTS

MODIFICATION: PROMPT_1
STATE: 0
FEATURE: NEW_PROMPTS
AUTHOR: Jane Doe
CREATION_DATE - TIME:      10/31/83 - 08.24.54
MODIFICATION_DATE - TIME: 10/31/83 - 08.24.54
MODIFICATION_DESCRIPTION: This adds a prompt for parameter
                          NEW_DECK.
MODIFICATION: PROMPT_2
STATE: 0
FEATURE: NEW_PROMPTS
AUTHOR: Jane Doe
CREATION_DATE - TIME:      11/05/83 - 13.29.04
MODIFICATION_DATE - TIME: 11/06/83 - 09.46.15
MODIFICATION_DESCRIPTION: This adds a prompt for parameter
                          OLD_DECK.

 Number of modifications associated with this feature: 2
```

DISPLAY_FEATURE_LIST SCU Subcommand

Purpose Lists the features in the source library.

Format

DISPLAY_FEATURE_LIST or
DISFL
OUTPUT=file
DISPLAY_OPTIONS=keyword
STATUS=status variable

Parameters

OUTPUT or *O*

File on which the display is written. You can specify a file position as part of the file name. If OUTPUT is omitted, file $OUTPUT is used.

DISPLAY_OPTIONS or *DO*

Specifies the information listed. Options are:

BRIEF (B)

Lists only the feature names.

FULL (F)

Lists the feature names and the names of the modifications that belong to each feature.

If DISPLAY_OPTIONS is omitted, BRIEF is used.

Remarks

- Features are listed alphabetically.
- To add a feature, create a modification that belongs to the feature. Once created, a feature name cannot be deleted from the feature list. If the feature list contains an unused feature name, you can enter an EXTRACT_SOURCE_LIBRARY command to remove the unused feature name from the result library.
- The feature list of the new library includes only those features with which modifications in the new library are associated and which have not been explicitly excluded by selection criteria commands.

- The DISPLAY_FEATURE_LIST subcommand is valid within an editing session started by an EDIT_DECK subcommand. It is also valid within a selection criteria file if prefixed with the slant character (/DISPLAY_FEATURE_LIST).
- For more information, see the NOS/VE Source Code Management manual.

Examples The following subcommand lists the features in the working library.

```
sc/display_feature_list
NEW_PROMPTS                    NEW_RESPONSE
```

DISPLAY_GROUP
SCU Subcommand

Purpose Lists the decks belonging to a group.

Format

DISPLAY_GROUP or
DISG
GROUP=name
ALTERNATE_BASE=list of file
OUTPUT=file
DISPLAY_OPTIONS=keyword
STATUS=status variable

Parameters

GROUP or **G**

Group name. This parameter is required.

ALTERNATE_BASE or *ALTERNATE_BASES* or *AB*

Optional list of one or more additional source libraries from which decks are listed if they belong to the group.

OUTPUT or *O*

File on which output is written. You can specify a file position as part of the file name. If OUTPUT is omitted, file $OUTPUT is used.

DISPLAY_OPTIONS or *DO*

Specifies the information listed. Options are:

BRIEF (B)

Lists only the deck names.

FULL (F)

Lists the deck names and the information in each deck header.

If DISPLAY_OPTIONS is omitted, BRIEF is used.

Remarks

- If you specify one or more alternate base libraries, DISPLAY_GROUP combines their group and deck lists with the working library group and deck lists for the duration of the subcommand.
- You can change the group to which a deck belongs with the CHANGE_DECK subcommand.
- The DISPLAY_GROUP subcommand is valid within an editing session started by an EDIT_DECK subcommand.
- For more information, see the NOS/VE Source Code Management manual.

Examples The following subcommand lists the decks in the group SECTION1.

```
sc/display_group section1
Decks associated with group SECTION1

FORTRAN_TEXT                FORTRAN_TEXT_II
FORTRAN_TEXT_III
```

DISPLAY_GROUP_LIST SCU Subcommand

Purpose Lists the groups in the library.

Format

DISPLAY_GROUP_LIST or
DISGL
ALTERNATE_BASE=list of file
OUTPUT=file
DISPLAY_OPTIONS=keyword
STATUS=status variable

Parameters

ALTERNATE_BASE or *ALTERNATE_BASES* or *AB*

Optional list of one or more libraries whose groups are listed with those of the base library.

OUTPUT or *O*

File on which the display is written. You can specify a file position as part of the file name. If OUTPUT is omitted, file $OUTPUT is used.

DISPLAY_OPTIONS or *DO*

Specifies the information listed. Options are:

BRIEF (B)

Lists only the group names.

FULL (F)

Lists the group names and the decks in each group.

If DISPLAY_OPTIONS is omitted, BRIEF is used.

Remarks

- Groups are listed alphabetically.
- If you specify one or more alternate base libraries, DISPLAY_GROUP_LIST combines their group and deck lists with the working library group and deck lists for the duration of the subcommand.
- To add a group, create a deck that belongs to the group. Once created, a group name cannot be deleted from the group list. If the group list contains an unused group name, you can enter an EXTRACT_SOURCE_LIBRARY command to remove the unused group name from the result library. The group list of the new library includes only those groups to which decks in the new library belong and which have not been explicitly excluded by selection criteria commands.
- The DISPLAY_GROUP_LIST subcommand is valid within an editing session started by an EDIT_DECK subcommand.
- For more information, see the NOS/VE Source Code Management manual.

Examples The following subcommand lists the groups on the working library and on library MY_LIB.

```
sc/display_group_list alternate_base=my_lib
SECTION1                              SECTION2
SECTION3
```

DISPLAY_LIBRARY
SCU Subcommand

Purpose Displays the library header of the working library.

Format

DISPLAY_LIBRARY or
DISL
OUTPUT=file
DISPLAY_OPTIONS=keyword
STATUS=status variable

Parameters

OUTPUT or *O*

File on which the display is written. You can specify a file position as part of the file name. If OUTPUT is omitted, file $OUTPUT is used.

DISPLAY_OPTIONS or *DO*

Specifies the information listed. Options are:

BRIEF (B)

Lists only library header information.

FULL (F)

Lists library header information and the names of the decks, groups, modifications, and features in the working library.

If DISPLAY_OPTIONS is omitted, BRIEF is used.

Remarks

- Besides the library header fields, DISPLAY_LIBRARY can also display the deck list, group list, modification list, and feature list of the working library.
- You can change the content of fields in the working library header with a CHANGE_LIBRARY subcommand. To reference a field in the library header, use the SCU function $LIBRARY_HEADER.

- The DISPLAY_LIBRARY subcommand is valid within an editing session started by an EDIT_DECK subcommand. It is also valid within a selection criteria file if prefixed with the slant character (/DISPLAY_LIBRARY).
- For more information, see the NOS/VE Source Code Management manual.

Examples The following subcommand displays the contents of the working library header.

```
sc/display_library
BASE=:nve.intve.scu.source_library

LIBRARY: SOURCE_CODE_UTILITY
VERSION: BUILD_12609
SCU_VERSION: 86133
LIBRARY_FORMAT_VERSION: V1.1
CHANGE_COUNTER: 394
LIBRARY_DESCRIPTION:  This library contains the source for
SOURCE_CODE_UTILITY (SCU) and associated SCL procedures.
CREATION_DATE - TIME:      07/31/81 - 13:15:43
MODIFICATION_DATE - TIME: 06/10/86 - 22:33:17
KEY: *
LAST_USED_DECK: SCP$GET_DEFAULT_RESOURCES
LAST_USED_MODIFICATION: SCB6134
COUNTS
 DECKS: 1237     MODS: 719     GROUPS: 41   FEATURES: 246
```

DISPLAY_MODIFICATION SCU Subcommand

Purpose Displays one or more modification headers.

Format

DISPLAY_MODIFICATION or
DISPLAY_MODIFICATIONS or
DISM
MODIFICATION=list of name or *keyword*
DECK=name or *keyword*
OUTPUT=file
DISPLAY_OPTIONS=keyword
STATUS=status variable

Parameters *MODIFICATION* or *MODIFICATIONS* or *M*

Modifications to be displayed. You can specify a list of one or more names, a list of one or more ranges, or the keyword ALL. ALL specifies all modification descriptions in the working library. If MODIFICATION is omitted, the last used modification is displayed.

DECK or *D*

Indicates whether the displayed information should apply to only the specified deck or to all decks. ALL specifies all decks in the working library. If DECK is omitted, ALL is used.

OUTPUT or *O*

File on which the display is written. You can specify a file position as part of the file name. If OUTPUT is omitted, file $OUTPUT is used.

DISPLAY_OPTIONS or *DO*

Specifies the information displayed. Options are:

BRIEF (B)

Displays the modification header only.

FULL (F)

Displays the modification header and the sequence of editing commands and inserted text that would produce the modification changes.

If DISPLAY_OPTIONS is omitted, BRIEF is used.

Remarks

- The DISPLAY_MODIFICATION subcommand is valid within an editor session started by an EDIT_DECK subcommand. It is also valid within a selection criteria file if prefixed with the slant character (/DISPLAY_MODIFICATION).
- For more information, see the NOS/VE Source Code Management manual.

Examples The following subcommand displays the modification MOD_4 description and changes.

```
sc/display_modification modification=mod_4 ..
sc../display_options=f
MODIFICATION: MOD_4
STATE: 0
FEATURE:
AUTHOR: Sam Spade
CREATION_DATE - TIME:      02/23/83 - 13:09:26
MODIFICATION_DATE - TIME: 02/24/83 - 08:14:01
MODIFICATION_DESCRIPTION: Fourth example modification
Text lines altered by modification MOD_4
SELECT_DECK X
INSERT_LINES P=BEFORE IL=FIRST UNTIL='///END\\\'
      do 10 i=1,10
   10 i = i+1
///END\\\
INSERT_LINES P=AFTER IL=MOD_3.2 UNTIL='///END\\\'
  100 i = i+100
///END\\\
SELECT_DECK Y
INSERT_LINES P=BEFORE IL=FIRST UNTIL='///END\\\'
*copyc z
///END\\\
```

DISPLAY_MODIFICATION_LIST SCU Subcommand

Purpose Lists all modifications in the working library.

Format **DISPLAY_MODIFICATION_LIST** or **DISML**

OUTPUT=file
DISPLAY_OPTIONS=keyword
STATUS=status variable

Parameters

OUTPUT or *O*

File on which the display is written. You can specify a file position as part of the file name. If OUTPUT is omitted, file $OUTPUT is used.

DISPLAY_OPTIONS or *DISPLAY_OPTION* or *DO*

Specifies the information listed.

ALPHABETIC (A)

Modifications are in alphabetical order.

CHRONOLOGICAL (C)

Modifications are ordered by date and time with the oldest modification first.

If DISPLAY_OPTIONS is omitted, ALPHABETIC is used.

Remarks

- To add a modification to the list, enter a CREATE_MODIFICATION subcommand. To remove a modification from the list, enter a DELETE_MODIFICATION subcommand.
- The DISPLAY_MODIFICATION_LIST subcommand is valid within an editing session started by an EDIT_DECK subcommand. It is also valid within a selection criteria file if prefixed with the slant character (/DISPLAY_MODIFICATION_LIST).
- For more information, see the NOS/VE Source Code Management manual.

Examples

The following subcommand lists all modifications in the working library.

```
sc/display_modification_list
MOD_1      MOD_2      MOD_3      MOD_4
```

EDIT_DECK
SCU Subcommand

Purpose Begins an editing session within an SCU session.

Format

EDIT_DECK or
EDID or
EDIT_LIBRARY or
EDIL
DECK=name
MODIFICATION=name
INPUT=file
OUTPUT=file
PROLOG=file
DISPLAY_UNPRINTABLE_CHARACTERS=boolean
STATUS=status variable

Parameters

DECK or *D*

Deck to be edited first.

NOTE

If the deck does not exist, it is created. If you have never entered a deck name on a DECK parameter, this parameter is required.

If DECK is omitted, the editing session begins with the last deck used.

To begin the editing session without selecting a deck, specify NONE on the DECK parameter.

MODIFICATION or *M*

Modification to which changes made during the editing session belong. For you to edit a deck using an existing modification, the modification must be in its initial state, state 0. If the modification does not already exist, it is created.

If MODIFICATION is omitted, the last modification is used. If you have never created a modification, this parameter is required.

INPUT or *I*

File from which commands are read. If INPUT is omitted, $COMMAND is used.

OUTPUT or *O*

File to which the display is written. If OUTPUT is omitted, file $OUTPUT is used. ($OUTPUT is usually connected to the terminal.)

PROLOG or *P*

File the system executes when you start an editing session. If PROLOG is omitted, file $USER.SCU_EDITOR_PROLOG is used.

DISPLAY_UNPRINTABLE_CHARACTERS or *DUC*

Specifies whether unprintable ASCII characters in the range 0 to 31 and 127 are replaced by mnemonics in the file. Options are:

TRUE

Unprintable characters are replaced by mnemonics, preceded by a less than symbol and followed by a greater than symbol, according to the ASCII character set.

FALSE

Unprintable characters are replaced by a single space and a warning message is issued if they are encountered. If the file is written when you exit the editing session, the mapping to spaces is written to the file.

If TRUE is specified, the mnemonics are replaced by the ASCII characters when the file is replaced. If DISPLAY_UNPRINTABLE_CHARACTERS is omitted, FALSE is used.

Remarks

- You can specify the deck to be edited with the DECK parameter. If you specify NONE on the DECK parameter, you must enter a deck selection subcommand before entering subcommands to change text.

- This subcommand adds an entry containing the EDIT_FILE utility subcommands to the NOS/VE subcommand list; the name of the entry is SCU_EDIT.

- If the interaction style you selected is SCREEN, the session occurs in full screen mode. The command CHANGE_INTERACTION_STYLE selects interaction modes.

- All editing subcommands and the deck selection subcommands that are available within the EDIT_FILE utility are described in the NOS/VE File Editor manual.

- The EDIT_FILE utility uses the tab columns specified in the deck header.

- Once you have started an editing session with an EDIT_DECK subcommand, you can then use an EDIT_FILE subcommand to edit a file.

- To discard decks that were created unintentionally, enter:

```
end_deck write_deck=false
```

- Once you have entered the SCU EDIT_DECK subcommand, you can enter the EDIT_DECK subcommand to edit other decks. This subcommand has only a DECK parameter.

- To change modifications, you must stop editing and enter the EDIT_DECK SCU subcommand specifying a different modification.

- The mnemonics that appear when DISPLAY_UNPRINTABLE_CHARACTERS=TRUE will be enclosed in less than and greater than symbols. For example, the mnemonic for the ASCII character 0 is NUL. This mnemonic appears on the terminal screen as follows: <NUL>

- For more information, see the NOS/VE Source Code Management manual.

Examples The following subcommand begins an editing session in line mode. All text changes belong to the new modification MOD_1.

```
sc/edit_deck modification=mod_1
sce/
```

The following is the header written on the output file if the EDIT_DECK subcommand is entered in batch mode.

```
EDITOR                                    08:39:10    PAGE 1
1986-07-09    NOS/VE SOURCE CODE UTILITY V1.1 86163
BASE=:nve.intve.scu.source_library.316
Begin editing deck SCM$SCU
```

END_LIBRARY
SCU Subcommand

Purpose Ends the interaction with the current working library. Another library can then be specified as the working library.

Format

END_LIBRARY or
ENDL
WRITE_LIBRARY=boolean
STATUS=status variable

Parameters *WRITE_LIBRARY* or *WL*

Specifies whether the working library should be written to the result file. The result file is specified in the CREATE_LIBRARY or USE_LIBRARY subcommand. If no result file was specified and you indicate that the working library should be written to the result file, then the library is written to file SOURCE_LIBRARY. If WRITE_LIBRARY is omitted, TRUE is used.

Remarks After entering the END_LIBRARY subcommand, you can work on another library by specifying either the USE_LIBRARY or CREATE_LIBRARY subcommand.

- After entering the END_LIBRARY subcommand, you can work on another library by specifying either the USE_LIBRARY or CREATE_LIBRARY subcommand.
- For more information, see the NOS/VE Source Code Management manual.

Examples The following example ends the association with the current working library. The library is written if changes have been detected by the $LIBRARY_MODIFIED function. Another library is then accessed by the USE_LIBRARY subcommand.

```
sc/end_library write_library=$library_modified
sc/use_library base=my_library result=new_library
```

$ERRORS_FILE
SCU Function

Purpose Returns the file to which intermediate diagnostic messages are written.

Format **$ERRORS_FILE**

Parameters None.

Remarks For more information, see the NOS/VE Source Code Management manual.

Examples The following command displays the current value of the file to which intermediate diagnostic messages are written.

```
/scu
sc/set_list_options errors=$user.my_error_file
sc/display_value $errors_file
$USER.MY_ERROR_FILE
```

EXCLUDE_DECK
Selection Criteria Subcommand

Purpose Explicitly excludes one or more decks.

Format **EXCLUDE_DECK** or
EXCLUDE_DECKS or
EXCD
DECK=list of range of name
STATUS=status variable

Parameters **DECK** or **DECKS** or **D**

Decks to be excluded. This parameter is required.

Remarks For more information, see the NOS/VE Source Code Management manual.

Examples The following sequence extracts modules from base library $USER.MY_LIBRARY using selection criteria commands. The extracted modules are then written to $USER.PART_OF_MY_LIBRARY.

```
/extract_source_library base=$user.my_library ..
../result=$user.part_of_my_library ..
../interlock=none selection_criteria=command
scc/include_group group1
scc/exclude_deck unwanted
scc/quit
```

The command sequence extracts all decks belonging to group GROUP1 except deck UNWANTED. When selection criteria entry has ended, the result is written on $USER.PART_OF_MY_LIBRARY.

EXCLUDE_FEATURE
Selection Criteria Subcommand

Purpose Explicitly excludes modifications belonging to one or more features.

Format

EXCLUDE_FEATURE or
EXCLUDE_FEATURES or
EXCF
FEATURE=list of name
STATE=integer
STATUS=status variable

Parameters

FEATURE or **FEATURES** or **F**

Features to be excluded. This parameter is required.

STATE or *S*

Maximum state (from 0 through 4) of modifications excluded. All modifications whose state is less than or equal to this value are excluded. If STATE is omitted, all modifications belonging to the feature are excluded.

Remarks

- This command is not valid for an EXTRACT_SOURCE_LIBRARY subcommand that sets an interlock.
- For more information, see the NOS/VE Source Code Management manual.

Examples

The following sequence extracts new source library $USER.MY_RESULT from the library on file $USER.MY_LIBRARY.

```
/extract_source_library decks=all ..
../base=$user.my_library ..
../result=$user.my_result ..
../interlock=none selection_criteria=command
scc/exclude_feature new_prompts
scc/quit
/
```

The sequence extracts all decks from the source library. However, it omits all lines of text belonging to modifications associated with the feature NEW_PROMPTS. It omits the feature NEW_PROMPTS from the feature list of the new library and the modifications associated with NEW_PROMPTS from the modification list.

EXCLUDE_GROUP
Selection Criteria Subcommand

Purpose

Explicitly excludes the decks belonging to one or more groups.

Format

EXCLUDE_GROUP or
EXCLUDE_GROUPS or
EXCG
GROUP=list of name
COMBINATION=keyword
STATUS=status variable

Parameters **GROUP** or **GROUPS** or **G**

Groups to be excluded. This parameter is required.

COMBINATION or *C*

Indicates whether the decks excluded must belong to one or all specified groups. Options are:

ANY

Excluded decks must belong to at least one of the specified groups.

ALL

Excluded decks must belong to all the specified groups.

If COMBINATION is omitted, ANY is used.

Remarks For more information, see the NOS/VE Source Code Management manual.

Examples The following subcommand sequence expands all decks on the working library except those belonging to group SECTION_1.

```
sc/expand_deck decks=all selection_criteria=command
scc/exclude_group group=section_1
scc/quit
```

EXCLUDE_LIBRARY
Selection Criteria Subcommand

Purpose Excludes decks found on one or more alternate base libraries. Although the command prevents you from selecting decks from specified libraries, COPY and COPYC directives processed by an EXPAND_DECK subcommand can still copy decks from the specified libraries.

Format

EXCLUDE_LIBRARY or
EXCLUDE_LIBRARIES or
EXCL
ALTERNATE_BASE=list of file
STATUS=status variable

Parameters **ALTERNATE_BASE or ALTERNATE_BASES or AB**

Source library files whose decks are excluded. The files must be a subset of the libraries specified on the ALTERNATE_BASE parameter of the subcommand. This parameter is required.

Remarks

- The EXCLUDE_LIBRARIES subcommand allows you to specify source libraries on the ALTERNATE_BASE parameter of the EXPAND_DECK subcommand that are to be used only for decks copied by COPY and COPYC directives. No other decks on the excluded library are expanded.
- For more information, see the NOS/VE Source Code Management manual.

Examples The following subcommand sequence expands all decks on the working library. Decks are copied from the library on file COMMON_LIBRARY if referenced by COPY or COPYC directives in the text.

```
sc/expand_decks decks=all alternate_base=..
sc../common_library selection_criteria=command
scc/exclude_library alternate_base=common_library
scc/quit
```

EXCLUDE_MODIFICATION
Selection Criteria Subcommand

Purpose Explicitly excludes one or more modifications.

Format

EXCLUDE_MODIFICATION or
EXCLUDE_MODIFICATIONS or
EXCM
MODIFICATION=list of name
STATUS=status variable

Parameters **MODIFICATION or MODIFICATIONS or M**

Modifications to be excluded. This parameter is required.

Remarks

- This subcommand is not valid for an EXTRACT_SOURCE_LIBRARY subcommand that sets an interlock.
- If several modifications of the same line exist, it is possible for an expanded deck to contain two versions of the same line if the modification deactivating the original line is excluded from the expanded deck.

  For example, assume Line 1 Version 1 is introduced by modification A. Modification B deactivates and replaces that line with Line 1 Version 2. Then modification C deactivates and replaces Line 1 Version 2 with Line 1 Version 3. If the deck is expanded with modification B excluded, both the Line 1 Version 1 and Line 1 Version 3 will appear in the compile file because Line 1 Version 1 is no longer activated.
- For more information, see the NOS/VE Source Code Management manual.

Examples

The following subcommand sequence expands all text in decks DECK1 through DECK3 except those lines belonging to modifications MOD2 and MOD4.

```
sc/expand_decks decks=(deck1..deck3) ..
sc../selection_criteria=command
scc/exclude_modification (mod2,mod4)
scc/quit
```

EXCLUDE_STATE
Selection Criteria Subcommand

Purpose

Explicitly excludes all modifications whose state is not greater than that specified.

Format

EXCLUDE_STATE or
EXCS
STATE=integer
STATUS=status variable

Parameters

STATE or **S**

Maximum state (from 0 through 3) of the modifications excluded. This parameter is required.

Remarks

- This command is not valid for an EXTRACT_SOURCE_LIBRARY subcommand that sets an interlock.
- For more information, see the NOS/VE Source Code Management manual.

Examples

The following subcommand sequence extracts all text in deck DECK1 except those lines belonging to modifications with a 0 (zero) or 1 state.

```
sc/extract_decks deck=deck1 selection_criteria=command
scc/exclude_state 1
scc/quit
```

EXPAND_DECK SCU Subcommand

Purpose

Expands one or more decks. When the SCU expands a deck, it processes directives embedded in the source text and copies the expanded text to a separate compile file.

Format

EXPAND_DECK or
EXPAND_DECKS or
EXPD

DECK=list of range of name
COMPILE=file
DEBUG_AIDS=keyword
OUTPUT_SOURCE_MAP=file
SELECTION_CRITERIA=file
WIDTH=integer
LINE_IDENTIFIER=keyword
ALTERNATE_BASE=list of file
LIST=file
EXPANSION_DEPTH=integer
DISPLAY_OPTIONS=keyword
ORDER=keyword
STATUS=status variable

Parameters

DECK or *DECKS* or *D*

Decks to be expanded. You can specify a list of one or more names, a list of one or more ranges, or the keyword ALL. ALL specifies all decks in the working library and in any alternate base libraries specified on the ALTERNATE_BASE parameter. If DECK is omitted, the

last deck used is expanded. To prevent the last used deck from being expanded, specify NONE on the DECK parameter. In that case, SCU determines the decks expanded by the subcommands entered via the selection criteria file.

COMPILE or *C*

File on which the expanded text is written. You can specify a file position as part of the file name. If COMPILE is omitted, file COMPILE is used.

DEBUG_AIDS or *DA*

If this parameter is set to DT, screen debugging information is written to the file named by the OUTPUT_SOURCE_MAP parameter. If DEBUG_AIDS is set to NONE or is omitted, no debugging information is produced.

OUTPUT_SOURCE_MAP or *OSM*

Names a file to receive screen debugging information specified by the DEBUG_AIDS parameter. If the file is not named, the screen debugging information is written to a file named OUTPUT_SOURCE_MAP.

SELECTION_CRITERIA or *SC*

File from which selection criteria commands are read. You can specify a file position as part of the file name. To enter selection criteria commands interactively, specify COMMAND. If SELECTION_CRITERIA is omitted, no selection criteria processing is performed and the DECK parameter specifies which decks will be expanded.

WIDTH or *W*

Length of the expanded lines excluding line identifiers. If WIDTH is omitted, SCU uses the default line width from the header of each deck.

LINE_IDENTIFIER or *LI*

Line identifier placement. Options are:

RIGHT (R)

Line identifiers are placed to the right of the text.

LEFT (L)

Line identifiers are placed to the left of the text.

NONE

No line identifiers are placed on output lines.

If LINE_IDENTIFIER is omitted, SCU uses the default line identifier placement from the header of each deck.

ALTERNATE_BASE or *ALTERNATE_BASES* or *AB*

Optional list of one or more additional libraries to be searched for decks.

LIST or *L*

Listing file. You can specify a file position as part of the file name. Within an SCU session, if LIST is omitted, the listing file is the file specified on the SET_LIST_OPTIONS subcommand. Otherwise, the default is file $LIST.

EXPANSION_DEPTH or *ED*

Number of levels of COPY and COPYC directives to process. COPY and COPYC directives beyond the maximum expansion depth are expanded as text. If EXPANSION_DEPTH is omitted, COPY and COPYC directives are processed whenever they are encountered.

DISPLAY_OPTIONS or *DO*

Indicates whether the listing includes the library for each deck from which the deck was expanded. Options are:

BRIEF (B)

Does not list the decks or their library origins.

FULL (F)

Lists the library origin when more than one library is used.

If DISPLAY_OPTIONS is omitted, BRIEF is used.

ORDER or *O*

Indicates whether the decks are expanded in the order specified or in alphabetical order. Options are:

COMMAND (C)

Decks are expanded in the order specified on the DECK parameter and by selection criteria commands.

LIBRARY (L)

Decks are expanded in alphabetical order.

If ORDER is omitted, LIBRARY is used.

Remarks

- For each deck specified by the DECK parameter, the EXPAND_DECK subcommand checks the expand attribute to determine if it expands the deck. If the expand attribute is TRUE, it expands the deck. If the expand attribute is FALSE, it skips the deck and continues processing with the next specified deck.

- To expand a text file, use the EXPAND_FILE subcommand and the EXPAND_SOURCE_FILE command.

  In order for OUTPUT_SOURCE_MAP to correctly reflect the origin of the text of each deck, the deck must either be unmodified or have been written to a result library. If a deck is encountered whose only current source is on the working library and the result library is currently scheduled for an actual file, then the currently scheduled result library is logged in the output source map as the origin and an error status is issued. A WRITE_LIBRARY subcommand must be entered to copy all decks from the working library to an actual file.

  If $NULL was specified as the result library, an error status is issued and the attempt aborts. A WRITE_LIBRARY subcommand must be entered, naming the result library. Then the EXPAND_DECK subcommand can be reissued.

- You can specify the decks to be expanded by name on the DECK parameter or by selection criteria commands in the selection criteria file or both. SCU begins with the decks specified on the DECK parameter and then adds and removes decks as specified by selection criteria commands. It omits any decks whose expand attribute is FALSE.

- You can specify alternate base libraries with the ALTERNATE_BASE parameter. SCU begins searching for a deck in the working library. If the deck is not found, SCU searches the ALTERNATE_BASE libraries in the order that they appear in the specified list.

- The EXPANSION_DEPTH parameter can limit the levels of nested directives processed. If SCU reads a directive at a level beyond the maximum level processed, it expands the directive as text.

- The LINE_IDENTIFIER, WIDTH, and ORDER parameters affect how the expanded text is written on the compile file. The LINE_IDENTIFIER and WIDTH parameters can override the default values in the deck headers. The ORDER parameter allows you to specify the order that SCU writes the decks on the file. If LINE_IDENTIFIER is explicitly stated in the EXPAND_DECK command, then the file attribute STATEMENT_IDENTIFIER is set. If LINE_IDENTIFIER is not explicitly stated, the system assumes that the file contents of the decks are inconsistent and does not set STATEMENT_IDENTIFIER.

- The line width can be specified by the WIDTH parameter. If the line width for a deck is 0 (zero), EXPAND_DECKS writes each line as it is stored in the deck (no trailing blanks or truncation); a blank line, therefore, is written as a zero-length V record. If the line width for a deck is nonzero, EXPAND_DECKS writes each line using that width. Lines shorter than the width are padded with trailing blanks; lines longer than the width are truncated.

- SCU issues a warning message for those decks that cannot be expanded.

- For more information, see the NOS/VE Source Code Management manual.

Examples The following subcommand expands the text of deck FORTRAN_TEXT and writes the expanded text on file FORTRAN_INPUT.

```
sc/expand_deck fortran_text fortran_input ..
sc../display_options=full alternate_base=ftnlib
*=Deck was copied
 FORTRAN_TEXT
*FTN_IO                                FTNLIB
*FTN_FORM                              FTNLIB
```

EXPAND_FILE
SCU Subcommand

Purpose Expands a text file. When the system expands a file, it processes the directives embedded in the source text and copies the expanded text to a separate compile file.

Format

EXPAND_FILE or
EXPF
FILE=file
COMPILE=file
DEBUG_AIDS=keyword
INPUT_SOURCE_MAP=file
OUTPUT_SOURCE_MAP=file
SELECTION_CRITERIA=file
WIDTH=integer
LINE_IDENTIFIER=keyword
ALTERNATE_BASE=list of file
LIST=file
EXPANSION_DEPTH=integer
DISPLAY_OPTIONS=keyword
STATUS=status variable

Parameters

FILE or F

File to be expanded. This parameter is required.

COMPILE* or *C

File on which the expanded text is written. You can specify a file position as part of the file name. If COMPILE is omitted, file COMPILE is used.

DEBUG_AIDS or *DA*

If this parameter is set to DT, screen debugging information is written to the file named by the OUTPUT_SOURCE_MAP parameter. If DEBUG_AIDS is set to NONE or is omitted, no debugging information is produced.

INPUT_SOURCE_MAP or *ISM*

Names a file from which screen debugging information is copied for the file specified by the FILE parameter. The content of the input source map is the output source map that was generated when the content of the FILE was produced. If INPUT_SOURCE_MAP is omitted, the screen debugging information describes lines read from FILE as having that origin.

OUTPUT_SOURCE_MAP or *OSM*

Names a file to receive screen debugging information specified by the DEBUG_AIDS parameter. If OUTPUT_SOURCE_MAP is omitted, the screen debugging information is written to a file named OUTPUT_SOURCE_MAP.

SELECTION_CRITERIA or *SC*

File from which selection criteria subcommands are read. You can specify a file position as part of the file name. To enter selection criteria subcommands interactively, specify COMMAND. If SELECTION_CRITERIA is omitted, no selection criteria processing is performed.

WIDTH or *W*

Length of the expanded lines, excluding line identifiers. If WIDTH is omitted, SCU uses 0 (zero) for the default line width. A line width of 0 (zero) means that lines can be up to 256 characters (with no trailing blanks) when the file is expanded.

LINE_IDENTIFIER or *LI*

Line identifier placement.

RIGHT (R)

Line identifiers are placed to the right of the text.

LEFT (L)

Line identifiers are placed to the left of the text.

NONE

No line identifiers are placed on output lines.

If LINE_IDENTIFIER is omitted, NONE is used.

ALTERNATE_BASE or *ALTERNATE_BASES* or *AB*

Optional list of one or more additional libraries to be searched for decks.

LIST or *L*

Listing file. You can specify a file position as part of the file name. Within an SCU session, if LIST is omitted, the listing file is the file specified on the SET_LIST_OPTIONS subcommand. Otherwise, the default is file $LIST.

EXPANSION_DEPTH or *ED*

Number of levels of COPY and COPYC directives to process. COPY and COPYC directives beyond the maximum expansion depth are expanded as text. If EXPANSION_DEPTH is omitted, COPY and COPYC directives are processed whenever they are encountered.

DISPLAY_OPTIONS or *DO*

Indicates whether the listing includes the library for each deck from which the deck was expanded.

BRIEF (B)

Does not list the decks or their library origins.

FULL (F)

Lists the library origin when more than one library is used.

If DISPLAY_OPTIONS is omitted, BRIEF is used.

Remarks

- To expand a deck, use the EXPAND_DECK subcommand.
- To expand a file while not in an SCU session, use the EXPAND_SOURCE_FILE command.
- You can specify alternate base libraries with the ALTERNATE_BASE parameter. When SCU processes a COPY or COPYC directive, it first searches the deck list of the working library for the deck specified on the directive and then it searches the deck lists of the alternate base libraries in the order the libraries are listed on the ALTERNATE_BASE parameter.
- The EXPANSION_DEPTH parameter can limit the levels of nested directives processed. If SCU reads a directive at a level beyond the maximum level processed, it expands it as text.
- The LINE_IDENTIFIER, WIDTH, and ORDER parameters affect how the expanded text is written on the compile file.
- The line width can be specified by the WIDTH parameter. If the line width for a file or deck is 0 (zero), EXPAND_FILE writes each line as it is stored in the file or deck (no trailing blanks or truncation); a blank line, therefore, is written as a zero-length V record. If the line width for a file or a deck is nonzero, EXPAND_FILE writes each line using that width. Lines shorter than the width are padded with trailing blanks; lines longer than the width are truncated.
- For more information, see the NOS/VE Source Code Management manual.

Examples

The following subcommand expands the text of file NEW_TEXT and writes the expanded text on file COMPILE. The unique name given to the temporary deck created from file NEW_TEXT is $82 .. 17.

```
sc/expand_file new_text display_options=full
*=Deck was copied
 $821497P3S0002D19860305T110817 Working Library
```

EXTRACT_DECK
SCU Subcommand

Purpose Extracts one or more decks. Extracting a deck copies the deck text to another file without processing directives embedded in the text. No delimiter is written between extracted decks.

Format

EXTRACT_DECK or
EXTRACT_DECKS or
EXTD
DECK=list of range of name
SOURCE=file
SELECTION_CRITERIA=file
WIDTH=integer
LINE_IDENTIFIER=keyword
ALTERNATE_BASE=list of file
LIST=file
DISPLAY_OPTIONS=keyword
ORDER=keyword
EXPAND=boolean or *keyword*
DECK_DIRECTIVES_INCLUDED=boolean
STATUS=status variable

Parameters

DECK or *DECKS* or *D*

Decks to be extracted. You can specify a list of one or more names, a list of one or more ranges, or the keyword ALL. ALL specifies all decks in the working library and in any alternate base libraries specified on the ALTERNATE_BASE parameter. If DECK is omitted, the last used deck is extracted. To prevent the last used deck from being extracted, specify NONE on the DECK parameter. In that case, SCU determines the decks extracted by the subcommands entered via the selection criteria file.

SOURCE or *S*

File on which the extracted text is written. You can specify a file position as part of the file name. If SOURCE is omitted, file SOURCE is used.

SELECTION_CRITERIA or *SC*

File from which selection criteria commands are read. You can specify a file position as part of the file name. If SELECTION_CRITERIA is omitted, no selection criteria processing is performed, and the decks extracted are determined by the DECK parameter.

WIDTH or *W*

Length of the extracted lines, excluding line identifiers. If WIDTH is omitted, the default line width for each deck is used.

LINE_IDENTIFIER or *LI*

Line identifier placement. Options are:

RIGHT (R)

Line identifiers are placed to the right of the text.

LEFT (L)

Line identifiers are placed to the left of the text.

NONE

No line identifiers are placed on output lines.

If LINE_IDENTIFIER is omitted, the default line identifier placement for each deck is used.

ALTERNATE_BASE or *ALTERNATE_BASES* or *AB*

Optional list of one or more additional libraries to be searched for decks.

LIST or *L*

Listing file. You can specify a file position as part of the file name. Within an SCU session, if LIST is omitted, the listing file is the file specified on the SET_LIST_OPTIONS subcommand. Otherwise, the default is file $LIST.

DISPLAY_OPTIONS or *DO*

Indicates whether the listing includes the library for each deck from which the deck was extracted. Options are:

BRIEF (B)

Does not list the decks or their library origins.

FULL (F)

Lists the library origin when more than one library is used.

If DISPLAY_OPTIONS is omitted, BRIEF is used.

ORDER or *O*

Indicates whether the decks are extracted in the order specified or in alphabetical order. Options are:

COMMAND (C)

Decks are extracted in the order specified on the subcommand.

LIBRARY (L)

Decks are extracted in alphabetical order.

If ORDER is omitted, LIBRARY is used.

EXPAND or *E*

Indicates the required expand attribute for each deck extracted. Options are:

TRUE

Expand attribute must be TRUE.

FALSE

Expand attribute must be FALSE.

ALL

Expand attribute can be either TRUE or FALSE.

If EXPAND is omitted, ALL is used.

DECK_DIRECTIVES_INCLUDED or *DDI*

Indicates whether a DECK directive precedes each extracted deck on the source file. Options are:

TRUE

A DECK directive is written before each deck.

FALSE

No DECK directives are written.

If DECK_DIRECTIVES_INCLUDED is omitted, FALSE is used.

Remarks

- The EXTRACT_DECK subcommand has the same deck selection options as the EXPAND_DECK subcommand. You can select the decks extracted by name, by selection critieria, or by both. However, unlike the EXPAND_DECK subcommand, you can also choose whether to use the expand deck attribute to select the decks to be extracted. With the EXPAND parameter, you can choose to extract decks whose expand attribute is TRUE, FALSE, or either TRUE or FALSE.

- You can use the extracted text as the source text when creating new decks. To include a DECK directive before the source text of each deck, specify DECK_DIRECTIVES_INCLUDED=TRUE on the subcommand. Using the embedded DECK directives, the decks created using the source text file will have the same names and expand attributes as the original decks.

- The EXTRACT_DECK subcommand does not save any of the deck header information such as DECK_DESCRIPTION. You must re-enter this information manually when you add the deck to the new library.

- You can specify alternate base libraries with the ALTERNATE_BASE parameter. SCU first searches the deck list of the working library for the deck and then searches the deck lists of the alternate base libraries in the order the libraries are listed on the ALTERNATE_BASE parameter.

- The LINE_IDENTIFIER, WIDTH, and ORDER parameters affect how the extracted text is written on the source file. The LINE_IDENTIFIER and WIDTH parameters can override the default values in the deck headers. The ORDER parameter allows you to specify the order that SCU writes the decks on the file.

- The line width can be specified by the WIDTH parameter. If the line width for a deck is 0 (zero), EXTRACT_DECK writes each line as it is stored in the deck (no trailing blanks or truncation); a blank line, therefore, is written as a zero-length V record. If

the line width for a deck is nonzero, EXTRACT_DECKS writes each line using that width. Lines shorter than the width are padded with trailing blanks; lines longer than the width are truncated.

- For more information, see the NOS/VE Source Code Management manual.

Examples The following subcommand extracts the text of deck FORTRAN_TEXT and writes the text on file SOURCE.

```
sc/extract_deck fortran_text display_option=full
FORTRAN_TEXT                        SOURCE_LIBRARY
```

EXTRACT_MODIFICATION SCU Subcommand

Purpose Generates a sequence of EDIT_FILE utility subcommands (INSERT_LINES, DELETE_LINES, and REPLACE_LINES subcommands) that, if processed, would introduce the modification changes.

Format

EXTRACT_MODIFICATION or
EXTRACT_MODIFICATIONS or
EXTM
MODIFICATION=list of range of name
EDIT_COMMANDS=file
DECK=name
TERMINATING_DELIMITER=string
STATUS=status variable

Parameters

MODIFICATION or *MODIFICATIONS* or *M*

Modifications to be extracted. If MODIFICATION is omitted, the last used modification is extracted.

EDIT_COMMANDS or EC

File to which the text and editing commands are written. You can specify a file position as part of the file name. This parameter is required.

DECK or *D*

Indicates whether the extracted modification lines should apply to only the specified deck or to all decks. ALL specifies all decks. If DECK is omitted, ALL is used.

TERMINATING_DELIMITER or *TD*

Delimiter string used to mark the end of inserted text (from 1 to 31 characters). If TERMINATING_DELIMITER is omitted, '///END\\\' is used.

Remarks

- The EXTRACT_MODIFICATION subcommand writes the editing commands and inserted text that make up a modification on a file. EXTM does not save any of the modification header information such as the author name or feature name. You must re-enter this information manually when you add the modification to the new library.
- Before deleting a modification, you can use the EXTRACT_MODIFICATION subcommand to save the modification changes on a separate file. You could then reintroduce the modification by processing the editing commands on the file.
- The subcommands can also extract only the modification changes that apply to one or more decks in the working library. To do so, specify the decks on the DECK parameter.
- If more than one modification is specified on the EXTRACT_MODIFICATION subcommand, the sequence of subcommands generated, if executed, would produce the combined modification changes.
- The EXTRACT_MODIFICATION subcommand is valid within an editing session started by an EDIT_DECK subcommand, but the modification changes extracted do not include any changes made since you last started editing the deck.
- For more information, see the NOS/VE Source Code Management manual.

Examples

The following subcommand extracts modification MOD1 onto file SAVE_MOD1.

```
sc/extract_modification mod1 save_mod1
```

$FEATURE
SCU Function

Purpose Returns a boolean value indicating whether the specified name is recognized as a feature on the working library.

Format

$FEATURE
(name)

Parameters

name

Name of the feature to be found. This parameter is required.

Remarks For more information, see the NOS/VE Source Code Management manual.

Examples The following command assigns a boolean value to the SCL variable FEATURE_EXISTS, depending on whether FEATURE1 is recognized as a feature in the working library.

```
sc/feature_exists = $feature(feature1)
```

$FEATURE_LIST
SCU Function

Purpose Returns an array of strings listing the names of features on the working library.

Format **$FEATURE_LIST**

Parameters None.

Remarks

- The array is ordered the same as it is on the working library.
- When used inside selection criteria subcommand processing, $FEATURE_LIST reflects the current feature list to be written to the compile, result, or source file being produced.
- For more information, see the NOS/VE Source Code Management manual.

Examples The following command assigns an array of strings containing the names of features on the working library to the variable FEATURE_LIST.

```
sc/feature_list = $feature_list
```

$FEATURE_MEMBERS
SCU Function

Purpose Returns an array of strings listing the names of modifications on the working library that belong to the specified feature.

Format

$FEATURE_MEMBERS
(name)

Parameters

name

Name of the feature. This parameter is required.

Remarks

- The names in the array appear in the same order as the names in the modification list in the working library.
- For more information, see the NOS/VE Source Code Management manual.

Examples The following command assigns to the variable FEATURES_MEMBERS an array of strings containing the names of modifications on the working library that belong to the feature NEW_VERSION.

```
feature_members = $feature_members(new_version)
```

The following example returns an array of strings listing the names of modifications on the working library that belong to the feature FEATURE_NAME.

```
sc/fm=$feature_members(feature_name)
sc/for i=1 to $variable(fm,upper_bound) do
for/display_value fm(i)
for/forend
MOD1
MOD2
MOD3
MOD4
sc/
```

$FIRST_DECK
SCU Function

Purpose Returns the name of the first deck in the working library as a string value.

Format **$FIRST_DECK**

Parameters None.

Remarks

- All letters in the string returned are uppercase, even if the name was originally entered using lowercase letters.
- For more information, see the NOS/VE Source Code Management manual.

Examples The following command assigns the name of the first deck to the SCL variable FIRST_DECK.

```
sc/first_deck = $first_deck
```

$FIRST_MODIFICATION
SCU Function

Purpose Returns the name of the first modification in the library modification list as a string value.

Format **$FIRST_MODIFICATION**

Parameters None.

Remarks

- All letters in the string returned are uppercase, even if the name was originally entered using lowercase letters.
- The modification list is kept in alphabetical order.
- For more information, see the NOS/VE Source Code Management manual.

Examples The following command assigns the name of the first modification to the SCL variable FIRST_MOD.

```
sc/first_mod = $first_modification
```

$GROUP
SCU Function

Purpose Returns a boolean value indicating whether a name is recognized as a group in the working library.

Format

$GROUP
(name)

Parameters

name

Name of the group to be searched for on the working library. This parameter is required.

Remarks For more information, see the NOS/VE Source Code Management manual.

Examples The following command assigns a boolean value to the variable GROUP_EXISTS, indicating whether the group TEST exists on the working library.

```
sc/group_exists = $group(test)
```

$GROUP_LIST
SCU Function

Purpose Returns an array of strings giving the names of the groups on the working library.

Format **$GROUP_LIST**

Parameters None.

Remarks

- The array is ordered the same as it is on the working library.
- When used in selection criteria subcommand processing, $GROUP_LIST reflects the current group list to be written to the compile, result, or source file.
- For more information, see the NOS/VE Source Code Management manual.

Examples The following command assigns an array of strings containing the names of groups on the working library to the variable GROUP_LIST.

```
sc/group_list = $group_list
```

$GROUP_MEMBERS
SCU Function

Purpose Returns an array of strings giving the names of decks on the working library that belong to the specified group.

Format

$GROUP_MEMBERS
(name)

Parameters

name

Name of the group whose members are to be listed. This parameter is required.

Remarks

- The array is ordered the same as it is on the working library.
- For more information, see the NOS/VE Source Code Management manual.

Examples The following command assigns to the variable GROUP_MEMBERS an array of strings giving the names of decks on the working library that belong to the group TEST.

```
sc/group_members = $group_members(test)
```

INCLUDE_COPYING_DECKS
Selection Criteria Subcommand

Purpose Explicitly includes all decks that contain a COPY or COPYC directive that directly or indirectly copies one of the specified decks.

Format

INCLUDE_COPYING_DECKS or
INCCD
DECK=list of range of name
DECK_RESIDENCE=keyword
STATUS=status variable

Parameters **DECK or DECKS or D**

Decks copied by the included decks. This parameter is required.

DECK_RESIDENCE or DR

Specifies whether the decks specified on the DECK parameter reside either on the working library or on alternate base libraries used by the subcommand. Options are:

EXTERNAL

The decks do not reside on the libraries.

INTERNAL

The decks reside on the libraries.

If DECK_RESIDENCE is omitted, INTERNAL is used.

Remarks

- The INCLUDE_COPYING_DECKS subcommand allows you to expand or extract only those decks that reference the specified decks.
- For more information, see the NOS/VE Source Code Management manual.

Examples The following subcommand sequence expands all decks that copy deck COMMON1.

```
sc/expand_decks selection_criteria=command
scc/include_copying_decks deck=common1
scc/quit
```

INCLUDE_DECK
Selection Criteria Subcommand

Purpose Explicitly includes one or more decks.

Format

INCLUDE_DECK or
INCLUDE_DECKS or
INCD
DECK=list of range of name
STATUS=status variable

Parameters **DECK or DECKS or D**

Decks to be included. This parameter is required.

Remarks

- If a deck name in a deck list is in error, the subcommand is not executed.
- For more information, see the NOS/VE Source Code Management manual.

Examples The following subcommand sequence excludes all decks in group GROUP1, but includes deck WANTED even if it belongs to GROUP1.

```
sc/expand_decks decks=all selection_criteria=command
scc/exclude_group group1
scc/include_deck wanted
scc/quit
```

INCLUDE_FEATURE
Selection Criteria Subcommand

Purpose Includes all modifications belonging to one or more features.

Format

INCLUDE_FEATURE or
INCLUDE_FEATURES or
INCF
FEATURE=list of name
STATE=integer
STATUS=status variable

Parameters

FEATURE or **FEATURES** or **F**

Features to be included. This parameter is required.

STATE or *S*

Minimum state (0 through 4) of the modifications included. All modifications whose state is greater than or equal to the specified state are included. If STATE is omitted, all modifications belonging to the feature are included.

Remarks For more information, see the NOS/VE Source Code Management manual.

Examples The following subcommand sequence expands DECK1 through DECK5. It includes all modifications belonging to feature NEW_PROMPTS that have a state of 2, 3, or 4.

```
sc/expd decks=deck1..deck5 selection_criteria=command
scc/include_feature feature=new_prompts state=2
scc/quit
```

INCLUDE_GROUP
Selection Criteria Subcommand

Purpose Explicitly includes decks belonging to one or more groups.

Format

INCLUDE_GROUP or
INCLUDE_GROUPS or
INCG
GROUP=list of name
COMBINATION=keyword
STATUS=status variable

Parameters

GROUP or **GROUPS** or **G**

Groups to be included. This parameter is required.

COMBINATION or *C*

Indicates whether the decks included must belong to any or all specified groups. Options are:

ANY

Included decks must belong to at least one of the specified groups.

ALL

Included decks must belong to all of the specified groups.

If COMBINATION is omitted, ANY is used.

Remarks For more information, see the NOS/VE Source Code Management manual.

Examples The following command sequence extracts all decks belonging to group SECTION_1.

```
sc/extract_decks selection_criteria=command
scc/include_group group=section_1
scc/quit
```

INCLUDE_MODIFICATION
Selection Criteria Subcommand

Purpose Explicitly includes one or more modifications.

Format

INCLUDE_MODIFICATION or
INCLUDE_MODIFICATIONS or
INCM
MODIFICATION=**list of name**
STATUS=status variable

Parameters

MODIFICATION or **MODIFICATIONS** or **M**

Modifications to be included. This parameter is required.

Remarks For more information, see the NOS/VE Source Code Management manual.

Examples The following command sequence expands all text on deck DECK5 except those lines belonging to feature MY_CHANGES. However, lines belonging to modifications MOD2 and MOD5 are expanded even if the modifications are associated with feature MY_CHANGES.

```
sc/expand_deck deck=deck5 selection_criteria=command
scc/exclude_feature my_changes
scc/include_modifications (mod2,mod5)
scc/quit
```

INCLUDE_MODIFIED_DECKS
Selection Criteria Subcommand

Purpose Explicitly includes all decks that are modified by a specified feature or modification. Decks directly modified are always included. Decks which copy modified decks (directly or indirectly through chains of indirect references) can also be optionally included.

Format

INCLUDE_MODIFIED_DECKS or
INCLUDE_MODIFIED_DECK or
INCMD
FEATURES=list of range of name
MODIFICATIONS=list of range of name
INCLUDE_COPYING_DECKS=boolean
STATUS=status variable

Parameters *FEATURES* or *FEATURE* or *F*

Name of features to be included. If FEATURE is omitted, MODIFICATION must be specified.

MODIFICATIONS or *MODIFICATION* or *M*

Names of modifications to be included. If MODIFICATION is omitted, FEATURE must be specified.

INCLUDE_COPYING_DECKS or *ICD*

Specifies whether decks that copy modified decks should be included. If INCLUDE_COPYING_DECKS is omitted, decks that copy modified decks are not included.

Remarks For more information, see the NOS/VE Source Code Management manual.

Examples The following example includes all of the decks modified by the modification ACCOUNTING_FIXES and all the decks that copy modified decks.

```
scc/include_modified_decks feature=accounting_fixes ..
scc../include_copying_decks=true
```

INCLUDE_STATE
Selection Criteria Subcommand

Purpose Includes all modifications whose state is greater than or equal to that specified.

Format **INCLUDE_STATE** or
INCS
STATE=integer
STATUS=status variable

Parameters **STATE** or **S**

Minimum state (from 0 through 4) of the modifications included. All modifications whose state is greater than or equal to the specified value are included. This parameter is required.

Remarks For more information, see the NOS/VE Source Code Management manual.

Examples The following command sequence extracts all lines in DECK5 belonging to modifications whose state is 2, 3, or 4.

```
sc/extract_deck deck=deck5 selection_criteria=command
scc/include_state 2
scc/quit
```

$LAST_DECK SCU Function

Purpose Returns the name of the last deck on the working library as a string value.

Format **$LAST_DECK**

Parameters None.

Remarks

- All letters in the string returned are uppercase, even if the name was originally entered using lowercase letters.
- For more information, see the NOS/VE Source Code Management manual.

Examples The following command assigns the name of the last deck to the SCL string variable LAST_DECK.

```
sc/last_deck = $last_deck
```

$LAST_MODIFICATION SCU Function

Purpose Returns the name of the last modification in the library modification list as a string value.

Format **$LAST_MODIFICATION**

Parameters None.

Remarks

- All letters in the string returned are uppercase, even if the name was originally entered using lowercase letters.
- The modification list is kept in alphabetical order.
- For more information, see the NOS/VE Source Code Management manual.

Examples The following command assigns the name of the last modification to the SCL string variable LAST_MOD.

```
sc/last_mod = $last_modification
```

$LIBRARY_HEADER SCU Function

Purpose Returns the contents of any library header field.

Format

$LIBRARY_HEADER
(keyword)

Parameters

keyword

Name of the field in the library header. This parameter is required. The field name can be one of the following:

CHANGE_COUNTER (CC)

Number of changes made to the library. The value is returned as an integer.

CREATION_DATE (CD)

Date the library was created. The value is returned as a string (MM/DD/YY).

CREATION_TIME (CT)

Time the library was created. The value is returned as a string (HH.MM.SS).

DECK_COUNT (DC)

Number of decks in the library. The value is returned as an integer.

FEATURE_COUNT (FC)

Number of features in the library. The value is returned as an integer.

GROUP_COUNT (GC)

Number of groups in the library. The value is returned as an integer.

KEY (K)

Key character. The value is returned as a string of 1 character.

LAST_USED_DECK (LUD)

Last value given explicitly for the DECK parameter. The value is returned as an uppercase string.

LAST_USED_MODIFICATION (LUM)

Last value given explicitly for the MODIFICATION parameter. The value is returned as an uppercase string.

LIBRARY (L)

Library name. The value is returned as a string (all letters are uppercase).

LIBRARY_DESCRIPTION (LD)

Library description. The value is returned as a string.

LIBRARY_FORMAT_VERSION (LFV)

Library format version. The value is returned as a string of up to 4 characters.

MODIFICATION_COUNT (MC)

Number of modifications in the library, including the original modification names associated with deck creation. The value is returned as an integer.

MODIFICATION_DATE (MD)

Date the library was last changed. The value is returned as a string (MM/DD/YY).

MODIFICATION_TIME (MT)

Time the library was last changed. The value is returned as a string (HH.MM.SS).

SCU_VERSION (SV)

SCU version. The value is returned as a string of up to 9 characters.

VERSION (V)

Library version. The value is returned as a string of up to 256 characters.

Remarks For more information, see the NOS/VE Source Code Management manual.

Examples The following command assigns the number of decks in the working library to the SCL integer variable NUMBER_OF_DECKS.

```
sc/number_of_decks = $library_header(deck_count)
```

$LIBRARY_MODIFIED SCU Function

Purpose Returns a boolean value indicating whether the current working library has been modified.

Format **$LIBRARY_MODIFIED**

Parameters None.

Remarks

- The value for $LIBRARY_MODIFIED is set to FALSE whenever a $WRITE_LIBRARY command is entered. TRUE means there are changes on the current working library that are not recorded on an external file.
- For more information, see the NOS/VE Source Code Management manual.

Examples The following command assigns a boolean value to the SCL variable LIBRARY_CHANGED, depending on whether the current working library has been modified.

```
sc/library_changed = $library_modified
```

$LIST_FILE
SCU Function

Purpose Returns the default listing file for the LIST parameter on SCU subcommands.

Format **$LIST_FILE**

Parameters None.

Remarks For more information, see the NOS/VE Source Code Management manual.

Examples The following command displays the current value of the default listing file.

```
/scu
sc/set_list_options list=$user.fortran_list_file
sc/display_value $list_file
$USER.FORTRAN_LIST_FILE
```

$MODIFICATION
SCU Function

Purpose Returns a boolean value indicating whether the specified modification is in the working library.

Format **$MODIFICATION**
(name)

Parameters **name**

Name of the modification to be found. This parameter is required.

Remarks

- If you exclude the specified modification using a selection criteria command, SCU evaluates the $MODIFICATION function as FALSE.
- For more information, see the NOS/VE Source Code Management manual.

Examples The following command assigns a boolean value to the SCL variable MOD_EXISTS, depending on whether MOD1 is in the working library.

```
sc/mod_exists = $modification(mod1)
```

$MODIFICATION_HEADER
SCU Function

Purpose Returns the contents of any modification header field.

Format

$MODIFICATION_HEADER
(name
keyword)

Parameters

name

Name of the modification whose header field is returned. This parameter is required.

keyword

The field in the modification header. This parameter is required. Options are:

AUTHOR (A)

Modification author. The value is returned as a string of up to 256 characters.

CREATION_DATE (CD)

Date when the modification was created. The value is returned as a string (MM/DD/YY).

CREATION TIME (CT)

Time when the modification was created. The value is returned as a string (HH.MM.SS).

FEATURE (F)

Feature to which the modification belongs. The value is returned as a string.

MODIFICATION_DATE

Date when lines were last added to the modification. The value is returned as a string (MM/DD/YY).

MODIFICATION_DESCRIPTION (MD)

Modification description. The value is returned as an array of strings.

MODIFICATION TIME (MT)

Time when lines were last added to the modification. The value is returned as a string (HH.MM.SS).

STATE (S)

Current state of the modification. The value is returned as an integer.

Remarks For more information, see the NOS/VE Source Code Management manual.

Examples The following command assigns the state of modification MOD4 to the SCL integer variable CURRENT_STATE.

```
sc/current_state = $modification_header(mod4,state)
```

$MODIFICATION_LIST SCU Function

Purpose Returns an array of strings listing the names of modifications on the working library.

Format **$MODIFICATION_LIST**

Parameters None.

Remarks

- The array is ordered alphabetically, as it is on the working library.
- When used in selection criteria subcommand processing, $MODIFICATION_LIST reflects the current modification list to be written to the compile, result, or source file being produced.
- For more information, see the NOS/VE Source Code Management manual.

Examples The following command assigns an array of strings giving the names of modifications on the working library to the variable MODIFICATION_LIST.

```
sc/modification_list = $modification_list
```

$MODIFIED_DECKS
SCU Function

Purpose Returns an array of strings giving the names of decks on the working library affected by a specified modification.

Format

$MODIFIED_DECKS
(name)

Parameters

name

Name of the modification. This parameter is required.

Remarks

- The array is ordered the same as it is on the working library.
- For more information, see the NOS/VE Source Code Management manual.

Examples The following command assigns to the variable MODIFIED_DECKS an array of strings giving the names of decks on the working library affected by the modification TEST.

```
sc/modified_decks = $modified_decks(test)
```

$NEXT_DECK
SCU Function

Purpose Returns the name of the next deck as a string value.

Format

$NEXT_DECK
(name)

Parameters

name

Name of the deck whose successor is to be found. This parameter is required.

Remarks

- All letters in the string returned are uppercase, even if the name was originally entered using lowercase letters.
- For more information, see the NOS/VE Source Code Management manual.

Examples The following command assigns the name of the deck following DECK1 to the SCL string variable NEXT_DECK.

```
sc/next_deck = $next_deck(deck1)
```

$NEXT_MODIFICATION
SCU Function

Purpose Returns the name of the next modification in the library modification list as a string value.

Format

$NEXT_MODIFICATION
(name)

Parameters

name

Name of the modification whose successor is to be found. This parameter is required.

Remarks

- All letters in the string returned are uppercase, even if the name was originally entered using lowercase letters.
- For more information, see the NOS/VE Source Code Management manual.

Examples The following command assigns the name of the modification following MOD1 to the SCL string variable NEXT_MOD.

```
sc/next_mod = $next_modification(mod1)
```

QUIT
Selection Criteria Subcommand

Purpose Ends SELECTION_CRITERIA_COMMAND command processing.

Format

QUIT or
END or
QUI
STATUS=status variable

Remarks For more information, see the NOS/VE Source Code Management manual.

QUIT
SCU Subcommand

Purpose Ends an SCU session and optionally writes the working library to the result source library.

Format

QUIT or
END or
QUI
WRITE_LIBRARY=boolean
STATUS=status variable

Parameters *WRITE_LIBRARY* or *WL*

Indicates whether SCU should generate a result library from the working library.

TRUE

SCU generates a result library.

FALSE

SCU does not generate a result library.

If WRITE_LIBRARY is omitted, TRUE is used.

Remarks

- The QUIT subcommand indicates whether SCU should generate a result library from the working library. If a library is to be generated, SCU writes the result library on the result library file specified on a CREATE_LIBRARY or USE_LIBRARY subcommand at the beginning of the session. If a WRITE_LIBRARY subcommand specifies a different result library, SCU writes the result library on the file specified by the last WRITE_LIBRARY subcommand. If none of these subcommands are specified, the result library is written on file SOURCE_LIBRARY in your working catalog.

- If the result file is the same as the file named on the BASE parameter of the USE_LIBRARY subcommand, it is rewritten only when the result library has been modified.

- Refer to WRITE_LIBRARY and END_LIBRARY for other subcommands that write a result library.

- For more information, see the NOS/VE Source Code Management manual.

Examples The following subcommand ends an SCU session and generates a result library.

```
sc/quit true
```

The following sequence changes and rewrites the source library and then ends the SCU session.

```
/scu
sc/use_library $user.my_library
sc/change_deck deck=deck1 author='roger'
sc/quit
```

REPLACE_LIBRARY SCU Subcommand

Purpose Replaces decks on the working library with decks from one or more source libraries.

Format

REPLACE_LIBRARY or
REPLACE_LIBRARIES or
REPL
SOURCE_LIBRARY=list of file
LIST=file
DISPLAY_OPTIONS=keyword
ENFORCE_INTERLOCKS=boolean
STATUS=status variable

Parameters

SOURCE_LIBRARY or **SOURCE_LIBRARIES** or **SL**

List of one or more source library names. This parameter is required.

LIST or *L*

Listing file. You can specify a file position as part of the file name. SCU lists the source library origin of each deck in the working library. If LIST is omitted, the listing file is the file specified on the SET_LIST_OPTIONS subcommand. Otherwise, the default is file $LIST.

DISPLAY_OPTIONS or *DO*

Specifies the information listed. Currently, both of the following keywords produce the same listing.

BRIEF or B
FULL or F

If DISPLAY_OPTIONS is omitted, BRIEF is used.

ENFORCE_INTERLOCKS or *EI*

Indicates whether the interlocks must match before a deck can replace a base library deck. Options are:

TRUE

Interlocks must match.

FALSE

Interlocks need not match.

If ENFORCE_INTERLOCKS is omitted, FALSE is used.

Remarks

- REPLACE_LIBRARIES reads the source library deck lists in the order you specify the libraries on the command.
- After reading a deck name, REPLACE_LIBRARIES determines if the deck name is in the working library deck list. If the name is in the list, it replaces the deck in the working library with the deck from the source library. If the name is not in the list, the command does not add the deck to the working library, but it sends a warning message, stating that the deck cannot be replaced because it is not in the working library.
- If no decks could be merged because an exception occurred in each deck, an error status is returned and REPLACE_LIBRARY makes no change to the library.
- REPLACE_LIBRARIES lists the source library origin of each deck in the working library on the listing file.
- Decks, features, groups, and modifications are ordered alphabetically on the REPLACE_LIBRARIES result library.

- You can use this subcommand to merge decks from an extracted library with decks from the original library from which it was extracted to form a new library. You use this command if you do not want to add any new decks to the new library.

  If you set interlocks when you extracted the library, REPLACE_LIBRARY enforces the interlock if you specify ENFORCE_INTERLOCKS=TRUE in the subcommand. Interlock enforcement means that REPLACE_LIBRARY checks whether the original interlock value in the header of the extracted deck copy matches the subinterlock value in the header of the working library copy. If the values match, REPLACE_LIBRARY replaces the working library deck with the extracted deck; otherwise, it does not replace the working library deck.

- Key characters in source libraries that are added to the working library must match the key character in the working library. If the key characters do not match, SCU generates an error message.

- For more information, see the NOS/VE Source Code Management manual.

Examples The following subcommand replaces decks on the working library with decks from source library NEWLIB.

```
sc/replace_library newlib
```

```
DECKA                          SOURCE_LIBRARY
DECKB                          NEWLIB
DECKC                          NEWLIB
DECKD                          SOURCE_LIBRARY
```

$RESULT
SCU Function

Purpose Returns the result library file.

Format **$RESULT**

Parameters None.

Remarks

- The value of $RESULT is updated when a WRITE_LIBRARY subcommand is entered that specifies a result file.
- For more information, see the NOS/VE Source Code Management manual.

Examples The following command displays the current value of the result file.

```
/scu
sc/use_library base=$user.fortran_lib ..
sc../result=$user.new_fortran_lib
sc/display_value $result
$USER.NEW_FORTRAN_LIB
```

RETAIN_GROUP
Selection Criteria Subcommand

Purpose Retains from the list of decks currently selected, only those decks that are members of the specified group.

Format

RETAIN_GROUP or
RETAIN_GROUPS or
RETG
GROUP=list of name
COMBINATION=keyword
STATUS=status variable

Parameters

GROUP or **GROUPS** or **G**

Names of the groups to be retained. This parameter is required.

COMBINATION or *C*

Decks to be retained. Options are:

ANY

Decks will be retained if they are members of any of the groups specified by the GROUP parameter.

ALL

Decks will be retained if they are members of all of the groups specified by the GROUP parameter.

If COMBINATION is omitted, ANY is used.

Remarks For more information, see the NOS/VE Source Code Management manual.

Examples The following example retains the decks which are at the same time members of group CYBIL and group SCF$UNBOUND_UTILITY.

```
scc/retain_groups groups=(cybil,scf$unbound_utility) ..
scc../combination=all
```

SEQUENCE_DECK
SCU Subcommand

Purpose Sequences deck lines in released state (state 4).

Format

SEQUENCE_DECK or
SEQUENCE_DECKS or
SEQD
DECK=list of range of name
MODIFICATION=name or *keyword*
STATUS=status variable

Parameters

DECK or **DECKS** or **D**

Decks to be sequenced. You can specify a list of one or more names, a list of one or more ranges, or the keyword ALL. ALL specifies all decks in the working library. This parameter is required.

MODIFICATION or *M*

Modification name that is used in the line identifiers for resequenced lines. If the modification already exists, it must be in state 4.

You specify that the creation modification is to be used for each deck by specifying the keyword CREATION_MODIFICATION.

If MODIFICATION is omitted, the creation modification for each deck is used.

Remarks

- To sequence a deck, you must have authority 4 for the file. The creation modification for each sequenced deck must be in state 4.
- The subcommand only sequences lines belonging to modifications in state 4. Each sequenced line is assigned a new line identifier. The line identifier consists of the name of the specified modification and a sequence number. The sequence numbers are assigned in the order the lines appear within the source library.
- After sequencing, all sequenced lines belong to the specified modification. The maximum sequence number is 16,777,214.
- If a sequenced deck has its subinterlock set, SCU reports a warning message.
- For more information, see the NOS/VE Source Code Management manual.

Examples The following subcommand sequences all decks in the working library.

```
sc/sequence_deck decks=all
```

SEQUENCE_MODIFICATION SCU Subcommand

Purpose Sequences modification lines.

Format

SEQUENCE_MODIFICATION or
SEQUENCE_MODIFICATIONS or
SEQM
MODIFICATION=list of range of name
DECK=list of range of name
STATUS=status variable

Parameters **MODIFICATION or MODIFICATIONS or M**

Modifications to be resequenced. This parameter is required.

DECK or DECKS or D

One or more decks. You can specify a list of one or more names, a list of one or more ranges, or the keyword ALL. ALL specifies all decks in the working library. If DECK is specified, only the modification lines that apply to the specified decks are sequenced. If DECK is omitted, ALL is used.

Remarks

- The sequenced modifications must be in state 0 (zero).
- Before sequencing, the sequence numbers in the line identifiers of a modification are ordered as the lines were added to the modification. After sequencing, the sequence numbers in the line identifiers are ordered as the lines appear in the deck. The maximum sequence number is 16,777,214.
- If a sequenced deck has its interlock set, SCU sends a warning message.
- You can specify the DECK parameter to limit sequencing to lines in the specified decks.
- For more information, see the NOS/VE Source Code Management manual.

Examples The following subcommand sequences modification MOD5.

```
sc/sequence_modification mod5
```

SET_LIST_OPTIONS SCU Subcommand

Purpose Establishes a default for the LIST parameters on SCU subcommands. It also specifies the file to which intermediate diagnostic messages are written.

Format

SET_LIST_OPTIONS or
SETLO
LIST=file
ERRORS=file
STATUS=status variable

Parameters

LIST or *L*

Default listing file for the LIST parameter used on subsequent subcommands in an SCU session. You can specify a file position as part of the file name. If LIST is omitted, file $LIST is used.

ERRORS or *E*

Name of the file on which intermediate error messages are written. If ERRORS is omitted, file $ERRORS is used.

Remarks

- This subcommand specifies the default value for the LIST parameter on SCU subcommands. A file specified for a LIST parameter overrides this value.
- The functions $ERRORS_FILE and $LIST_FILE return the values specified for these files.
- For more information, see the NOS/VE Source Code Management manual.

Examples

The following subcommand causes file SCU_LIST to be used as the default value for the LIST parameter on subsequent subcommands. Intermediate error messages are written on file SCU_ERRORS.

```
sc/set_list_options list=scu_list errors=scu_errors
```

USE_LIBRARY
SCU Subcommand

Purpose

Specifies the base and result libraries for an SCU utility session. This subcommand also specifies where the QUIT, END_LIBRARY, and WRITE_LIBRARY subcommands write their results.

Format

USE_LIBRARY or
USEL
BASE=file
RESULT=file
STATUS=status variable

Parameters

BASE or *B*

Name of the source library copied as the initial working library for the session. The files specified by the BASE and RESULT parameters can be the same. If BASE is omitted, file SOURCE_LIBRARY in your working catalog is used.

RESULT or *R*

Name of the file on which the new source library is written by subsequent END_LIBRARY, WRITE_LIBRARY, or QUIT subcommands. The new source library can be written when either a QUIT, END_LIBRARY, or WRITE_LIBRARY subcommand is entered. The WRITE_LIBRARY subcommand can specify a different source library than that specified by the USE_LIBRARY subcommand. The files specified by the BASE and RESULT parameters can be the same. If RESULT is omitted, the file specified by the BASE parameter is used.

Remarks

- All subcommands in the session affect the same working library. The working library is initially a duplicate of the base library specified on the BASE parameter.
- If no USE_LIBRARY or CREATE_LIBRARY subcommand is issued before other subcommands during an SCU session, file SOURCE_LIBRARY is used for the base and result libraries.
- You must have read and execute permission on the base library. You must have read and write permission on the result library. If you only want to read the base library, specify $NULL as the result library.
- For more information, see the NOS/VE Source Code Management manual.

Examples

The following sequence begins an SCU session and intitializes the working library from file FSEWORK in your working catalog, assumed not to be $LOCAL. In this example, source libraries are written on the next cycle of file FSEWORK by subsequent END_LIBRARY, WRITE_LIBRARY, or QUIT subcommands.

```
/source_code_utility
sc/use_library base=fsework result=fsework.$next
```

The following sequence specifies $NULL as the result library. You can use this example to look at a source library, but not to change it.

```
/source_code_utility
sc/use_library ..
sc../$system.cybil.osf$program_interface result=$null
```

WRITE_LIBRARY SCU Subcommand

Purpose Generates a result library from the current state of the working library. It writes the result library on the file specified by the RESULT parameter.

Format

WRITE_LIBRARY or
WRIL
RESULT=file
STATUS=status variable

Parameters *RESULT* or *R*

File to which the result library is written. If RESULT is omitted, the file used is specified by the RESULT parameter of the CREATE_LIBRARY, previous WRITE_LIBRARY, or USE_LIBRARY subcommand. If RESULT is specified, that file name becomes the default for subsequent QUIT or WRITE_LIBRARY subcommands.

Remarks

- This subcommand allows you to generate more than one source library in an SCU session. This is done if you specify a file on the RESULT parameter. To create an empty library, refer to the CREATE_LIBRARY subcommand.

- The subcommand can save the contents of the working library at an intermediate state in case the system fails during the session. In this case, you can omit the RESULT parameter and use the result file you specified when you began the session. When you end the session, you can overwrite the intermediate library with the final result library.

- If the result file is the same as the file named on the BASE parameter of the USE_LIBRARY subcommand, the file is rewritten only if the working library has been modified.
- The END_LIBRARY and QUIT subcommands also generate a result library.
- Specifying RESULT changes the value of the $RESULT function to reflect the new file name.
- For more information, see the NOS/VE Source Code Management manual.

Examples The following subcommand writes an intermediate library to the result library file.

```
sc/write_library
```

Related Manuals A

The following lists the categories of manuals which relate to NOS/VE.

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.

Related Manuals A

All NOS/VE manuals and related hardware manuals are listed in table A-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 A-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 A-1, that manual is represented in the online Examples manual.

Table A-1. Related Manuals

Manual Title	Publication Number	Online Manuals[1]
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 A-1. Related Manuals *(Continued)*

Manual Title	Publication Number	Online Manuals[1]
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 A-1. Related Manuals *(Continued)*

Manual Title	Publication Number	Online Manuals[1]
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	SCU 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 A-1. Related Manuals *(Continued)*

Manual Title	Publication Number	Online Manuals[1]
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 A-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 A-1. Related Manuals *(Continued)*

Manual Title	Publication Number	Online Manuals[1]
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 A-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 A-1. Related Manuals *(Continued)*

Manual Title	Publication Number	Online Manuals[1]
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 A-1. Related Manuals *(Continued)*

Manual Title	Publication Number	Online Manuals[1]
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 A-1. Related Manuals *(Continued)*

Manual Title	Publication Number	Online Manuals[1]
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 A-1. Related Manuals *(Continued)*

Manual Title	Publication Number	Online Manuals[1]
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 A-1. Related Manuals *(Continued)*

Manual Title	Publication Number	Online Manuals[1]
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 A-1. Related Manuals *(Continued)*

Manual Title	Publication Number	Online Manuals[1]
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 A-1. Related Manuals *(Continued)*

Manual Title	Publication Number	Online Manuals[1]
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 A-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.

Comments (continued from other side)

Please fold on dotted line;
seal edges 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

NOS/VE Commands and Functions 60464018 G

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	
☐	☐	☐	Is the manual easy to read (print size, page layout, and so on)?
☐	☐	☐	Is it easy to understand?
☐	☐	☐	Does it tell you what you need to know about the topic?
☐	☐	☐	Is the order of topics logical?
☐	☐	☐	Are there enough examples?
☐	☐	☐	Are the examples helpful? (☐ Too simple? ☐ Too complex?)
☐	☐	☐	Is the technical information accurate?
☐	☐	☐	Can you easily find what you want?
☐	☐	☐	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.

Replacements for Old NOS/VE Commands

The following table 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.

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

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_PASSWORD	CHANGE_LOGIN_PASSWORD
SET_TERMINAL_ATTRIBUTES	CHANGE_TERMINAL_ATTRIBUTES
TERMINATE_PRINT	TERMINATE_OUTPUT

Command, Subcommand, and Control Statement Index

This index lists the commands, subcommands, and control statements described in this manual and the page on which each is described. Each subcommand entry is followed by the word "sub" and the abbreviation of the command that starts the utility session.

E

T

Functions Index

This index lists the functions described in this manual and the pa which each is described. Some functions are followed by the word "fun" abbreviation of the command utility in which the function can be used.

CONTROL DATA