NOS/VE
Source Code Management
Usage

60464313

CONTROL DATA

Command, Subcommand, and Directive Index

This list provides a quick reference to each command, subcommand, and directive description.

Function Index

NOS/VE

Source Code Management

Usage

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

Publication Number 60464313

Manual History

Revision	System Version	PSR Level	Date
A	1.0.2	–	October 1983
B	1.1.1	621	July 1984
C	1.1.3	644	October 1985
D	1.2.1	664	September 1986
E	1.2.2	678	April 1987
F	1.2.3	688	September 1987
G	1.3.1	700	April 1988

Revision G of this manual, printed April 1988, documents the management of source code for NOS/VE Version 1.3.1 at PSR level 700. The following command and functions have been added to the manual:

END_LIBRARY
$BASE
$ERRORS_FILE
$LIST_FILE
$RESULT

The following parameters have been added to the given command:

CHANGE_DECK	CLEAR_ORIGINAL_INTERLOCK CLEAR_SUB_INTERLOCK
CHANGE_LIBRARY	LAST_USED_DECK
	LAST_USED_MODIFICATION
EDIT_DECK	DISPLAY_UNPRINTABLE_CHARACTERS

Miscellaneous editorial and technical corrections have been made. This edition obsoletes all previous editions.

Contents

Figures

Tables

About This Manual

This manual describes the System Command Language (SCL), which provides the user interface to the CONTROL DATA® Network Operating System/Virtual Environment (NOS/VE).

This manual describes SOURCE_CODE_UTILITY, a development tool that organizes and maintains libraries of ASCII source code. Features include deck editing and extraction, conditional text expansion, modification state constraints, and use of the EDIT_FILE utility.

Audience

This manual is written for any NOS/VE user who uses or maintains libraries of source text. It assumes you are familiar with NOS/VE file concepts and the SCL command syntax and language. These concepts are described in the NOS/VE System Usage manual. The manual also assumes you are familiar with the EDIT_FILE utility described in the NOS/VE File Editor manual.

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 SC8 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 to SCU 1

Introduction to SCU 1

The Source Code Utility (SCU) is a NOS/VE command utility designed to store, organize, manipulate, and extract units of text. The Source Code Utility is designed primarily as a development tool for large systems or applications development groups. However, it can also be used by anyone who is responsible for maintaining source code libraries.

Although you can use it for any collection of text, SCU is primarily designed for source text. Source text is text input for a processor, such as program text for a compiler or procedure or job text for the System Command Language (SCL) interpreter. Source text is stored in libraries, which are NOS/VE files that have a unique format and structure.

You can also manipulate SCU libraries using the Professional Programming Environment (PPE), which provides a full-screen interface to SCU. To use the Professional Programming Environment, enter the command ENTER_PPE. PPE is a full-screen, object-oriented software development tool that coordinates the activities of large multi-person programming projects. Using PPE, you can create and delete decks and modifications, transmit and extract decks and modifications, expand and compile product source code, and maintain object and source libraries. More information is available in the Professional Programming Environment for NOS/VE manual.

This chapter explains how to use SCU and describes terminology, basic features, and the more advanced capabilities of SCU.

All references to commands and their parameters within this chapter use the complete command or parameter name. Command and parameter names often have both singular and plural forms and also abbreviated forms. All forms of the command and parameter names are included in the individual description of the command.

Terminology

Before attempting to learn SCU capabilities, you must understand the terms used for the different storage mechanisms and SCU units of data. The following paragraphs define units as they are commonly used.

Source Text Storage Units

Text is most usable when it is stored as units that can be accessed independently.

For example, program text is most usable when it is stored as individual compilation units. A compilation unit is a sequence of lines compiled as an independent unit, such as a CYBIL module or a FORTRAN program, subroutine, or function.

Similarly, text for individual procedures or jobs is most usable when it is stored so that each procedure or job can be accessed individually.

As illustrated in figure 1-1, SCU uses the following text entities to store and organize text units.

Line

Sequence of characters. SCU assigns a unique identifier to each line so you can reference it individually.

Deck

Collection of lines with a header describing the collection. For example, a deck could be a compilation unit, a procedure, or a job. You reference a deck by its name.

Source library

Collection of decks on a file with a header describing the collection. You reference a source library by the file on which it resides.

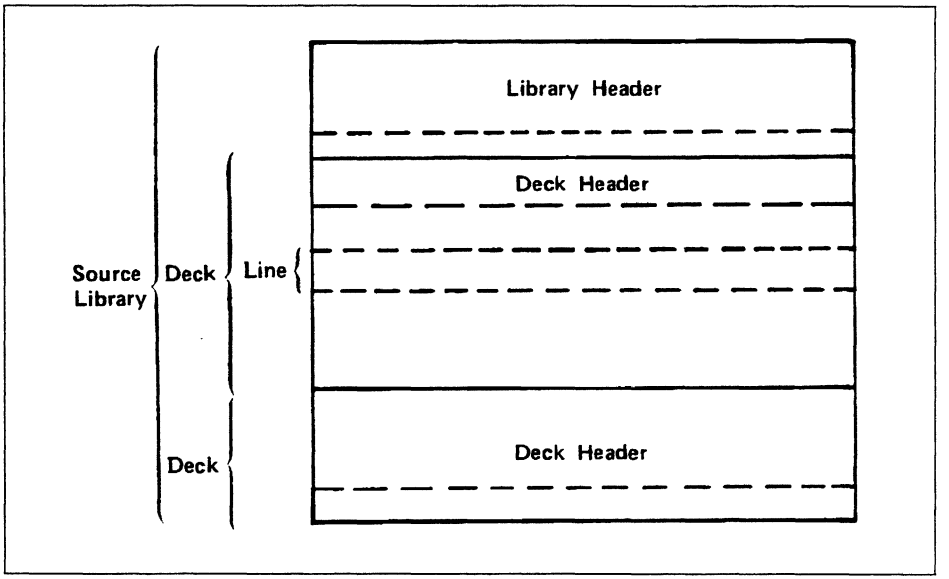


Figure 1-1. SCU Text Units

Source Text Change Record

The process of debugging a program usually involves changing the source text or input data, recompiling and executing the program, and determining how the program results changed. The process is then repeated. It is often important while debugging a program to maintain a record of each set of changes made.

SCU requires that each set of text changes (including additions and deletions) be associated with a modification name. You can later reference the set of changes by its modification name. A modification is defined as follows.

Modification

Each line in a deck belongs to a modification, and the modification name is part of the line identifier. All lines belonging to a modification can be referenced by that modification name. The modification can be deactivated and reactivated, as described later in this chapter.

Project Organization Units

A large programming project is often split into several subprojects, and each subproject could involve more than one compilation unit. SCU provides the following logical units to assist in project organization.

Feature

Collection of modifications. A modification can belong to only one feature. All modifications belonging to a feature can be referenced by the feature name. Modifications do not have to belong to a feature.

Group

Subset of decks within a library. A deck can belong to one group, more than one group, or no group. All decks belonging to a group can be referenced by the group name.

Modifications, features, and groups can be used for specifying the decks and lines to be processed. For certain commands, you can specify a separate file that lists the decks, groups, modifications, and features to be processed.

Entering Commands

An SCU subcommand is valid only within an SCU session. An SCU session begins when you enter the SOURCE_CODE_UTILITY command. The session ends when you enter a QUIT subcommand.

You use the EDIT_FILE utility to enter and change SCU source text. An EDIT_FILE session (also called an editing session) begins when you enter EDIT_DECK or the EDIT_FILE command within an SCU session.

The editing session ends when you enter QUIT. If you begin the editing session within an SCU session, you remain in the SCU session when you end the editing session.

For further information about the EDIT_FILE utility, refer to chapter 3 in this manual, and to the NOS/VE File Editor manual.

During a NOS/VE interactive terminal session, you can determine the valid commands by the input prompt that appears on the screen.

- / prompts you for a NOS/VE command.
- sc/ prompts you for an SCU subcommand or a NOS/VE command.
- scc/ prompts you for selection criteria subcommands. This prompt appears when you specify COMMAND for the SELECTION_CRITERIA parameter on an EXPAND_DECK, EXTRACT_DECK, or EXPAND_FILE subcommand or EXTRACT_SOURCE_LIBRARY or EXPAND_SOURCE_FILE command.
- sce/ indicates you are in an editing session started by an EDIT_DECK subcommand in line mode. You may enter an EDIT_FILE subcommand, an SCL command, or a valid SCU subcommand. (The individual SCU subcommand description indicates whether the subcommand is valid in an editing session.) If you are editing in screen mode, the prompt does not appear.
- ef/ indicates you are in an editing session started by an EDIT_FILE command in line mode (described in the NOS/VE File Editor manual). You are prompted for an EDIT_FILE subcommand or a NOS/VE command. If you are editing in screen mode, the prompt does not appear.

Figure 1-2 shows an example of an interactive session using these prompts.

`/source_code_utility`	Begins an SCU session.
`sc/create_library result=..` `sc../any_library`	Creates an empty library and specifies the result library.

`sc/edit_deck deck=init_array ..`
`sc../modification=firstmod`
`Begin editing deck INIT_ARRAY`
	Starts the editing session and creates a deck.

`sce/insert_lines`	Editing command. Inserts text in the deck.

`Enter text`
`?  program myprog`
`?  stop`
`?  end**`
	Text entry.

`sce/end`	Ends the editing session.
`sc/expand_deck deck=init_array`	Expands a deck and writes the expanded source text to the file COMPILE.
`sc/fortran input=compile`	Compiles the source text.
`sc/quit` `/`	Ends the SCU session and writes the result library.

Figure 1-2. Command Prompt Example

Creating a Source Library

To store text using SCU, you can either store it on an existing source library or create a new one. SCU can accept one existing source library as input, and it can overwrite that library. Or, alternatively, the new source library can be written as a new cycle of the existing source library file. A source library file contains decks, modifications, features, and groups that organize the text. This type of file has specific file structure and file content attributes that make it behave differently from other NOS/VE files. When you specify a source library file on a command, the system determines if the file you have entered has the correct attributes for a library file.

To use modification states and set interlocks, a source library must be a permanent file.

The following paragraphs describe how to create a new source library.

Base, Working, and Result Libraries

Within this manual, SCU processing is described in terms of a base library, a working library, and a result library as shown in figure 1-3. You specify a base library and a result library with the SCU command USE_LIBRARY.

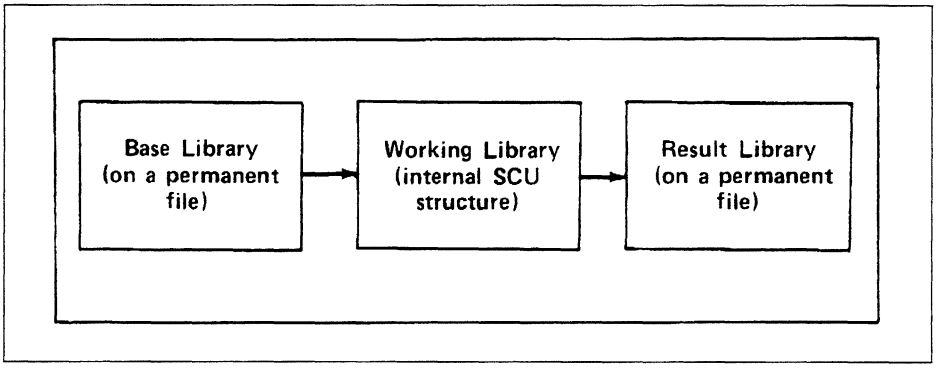


Figure 1-3. Source Library Processing

The base library is the starting point for the new library to be created. If you enter no commands to modify the new library, the new library is a duplicate of the base library.

The working library is the current state of the new library; it is the new library in progress. Initially, it is a duplicate of the base library. However, as you enter SCU subcommands, the content of the working library changes. SCU subcommands affect only the working library; they do not change the base library.

The result library is the new source library SCU writes.

The file SOURCE_LIBRARY in your working catalog is used for both base and result libraries during an SCU session if you do not enter a CREATE_LIBRARY or a USE_LIBRARY subcommand before other subcommands.

You must have read permission to the base library file and read and write permission to the result library file. (Write permission is the combination of modify, shorten, and append permissions.) For information on read and write permissions to files, refer to the NOS/VE System Usage manual.

You specify the base and result libraries for the session on the USE_LIBRARY subcommand. When you end the SCU session, changes will be saved on the result library file unless you specify otherwise.

The following is an example of an SCU session.

```
/source_code_utility
sc/use_library base=$user.my_library
sc/create_deck deck=new_deck modification=modif1 ..
sc../source=$user.new_deck_source
sc/quit
```

Because no result library was specified by USE_LIBRARY, the result library defaults to $USER.MY_LIBRARY. The CREATE_DECK subcommand creates a deck called NEW_DECK by using the contents of file $USER.NEW_DECK_SOURCE. The QUIT subcommand ends the utility session. The base library on file $USER.MY_LIBRARY is rewritten with the changed library.

Creating a Base Library

The first subcommand you usually enter in an SCU session is USE_LIBRARY. With it you select a base and a result library. The base library could be an existing source library or a source library extracted from an existing source library. Or you could omit the USE_LIBRARY subcommand and create a new empty source library with the CREATE_LIBRARY subcommand during SCU processing. The following are guidelines on creating a base library:

- Specify an existing source library as the base library if the new source library is to be a new version of the existing source library.
- Extract a source library when the new source library is to contain a subset of the decks on the base library. Use of an extracted source library is described later in this chapter.
- Create an empty source library if the new library is to contain all new decks.

To use an existing source library as the base library, you need not attach the library file. All you need to do is specify the file on the BASE parameter of the USE_LIBRARY subcommand, assuming you have read access permission to the source library file.

To create a new empty source library, enter a CREATE_LIBRARY subcommand after beginning an SCU session. The subcommand creates a new library that is used as the working library during this SCU session.

The file specified on the RESULT parameter of the CREATE_LIBRARY subcommand is used as the result file for the SCU session. You must have modify, shorten, and append permission to the file.

The following example allows you to change all modification states and set interlocks on source library file $USER.SOURCE_LIBRARY.

```
/create_file_permit file=$user.source_library ..
../group=user user=archie access_mode=(read modify) ..
../share_mode=none application_information='I4'
```

The I in the APPLICATION_INFORMATION parameter allows you to interlock decks. The 4 in the APPLICATION_INFORMATION parameter allows you to alter the state of all modifications. The NOS/VE command CREATE_FILE_PERMIT is described in the NOS/VE System Usage manual.

Creating a Deck

After you have determined the base library and result library files to use, you can store text as a deck on the result library. A deck can be created by using one of the following methods:

- Create a deck containing text copied from a file
- Create an empty deck and use the EDIT_FILE utility to enter text in the deck
- Create a deck implicitly by naming it on the EDIT_FILE command

The SCU subcommand CREATE_DECK creates a deck on a library. If you specify a file using the SOURCE parameter of the CREATE_DECK subcommand, SCU copies text from the file to the new deck. If you omit the SOURCE parameter, SCU creates an empty deck.

The SCU subcommand EDIT_DECK creates an empty deck if the specified deck does not exist. You must specify a deck name and a modification name if one has not been specified before.

Deck Name

The name you give a deck when you create it is the name by which it is later referenced. The name can be from 1 through 31 characters and must follow the SCL naming conventions as defined in appendix A; no other deck on the library may have the same name. The deck name can be specified on the EDIT_DECK subcommand, on the CREATE_DECK subcommand, or on DECK directives embedded in source text. SCU orders decks on a source library alphabetically by name (lowercase characters are converted to uppercase).

Modification Name

Besides a deck name, you can also specify a modification name on a CREATE_DECK subcommand. The modification name can only be from 1 to 9 characters long. It can name a new modification or a modification already existing on the library.

The modification used on the CREATE_DECK subcommand names the creation modification for the deck. It applies to all lines of text SCU copies to the deck during its creation. It can also apply to later editing of the deck if the same modification name is referenced. If you do not specify a modification name, the most recently specified name is used.

Each line identifier contains a modification name and a sequence number which SCU assigns to lines in the order it creates them. A line identifier has the following format.

```
modification_name.sequence_number
```

For example, suppose you enter the following command sequence to create a new source library and a new deck on the library.

```
/source_code_utility
sc/create_library result=$user.my_library
sc/collect_text output=in_file
ct?       program example
ct?       print *, 'Hello!'
ct?       stop
ct?       end
ct? **
sc/create_deck deck=new_deck modification=modif1 source=in_file
sc/quit
```

The SCU subcommand CREATE_LIBRARY creates an empty source library. The COLLECT_TEXT command enters text on a file named IN_FILE. The SCU subcommand CREATE_DECK creates a deck named NEW_DECK and copies the text on file IN_FILE to the deck. The QUIT subcommand ends the SCU session and, by default, writes the new source library on file $USER.MY_LIBRARY. The new library contains only one deck, NEW_DECK.

SCU assigns the following line identifiers to the lines in the new deck.

```
MODIF1      1       program example
MODIF1      2       print *, 'Hello!'
MODIF1      3       stop
MODIF1      4       end
```

Generating an Expanded Text File from a Deck

Text is stored in a deck for later access. For example, program text is accessed for compilation. After storing a compilation unit as a deck on a source library, you can generate an expanded text file for use as a compiler input file. Expanding a deck processes the directives embedded in the source text and copies the expanded text to a separate compile file.

Generating an expanded text file involves the following steps.

1. Selecting the decks to be expanded.
2. Expanding the text as stored by SCU, including processing of directives embedded in the text and writing the expanded text to a file.

The SCU subcommand EXPAND_DECK performs both steps.

The following example shows the creation and use of an input file for the FORTRAN compiler.

```
/scu
sc/use_library base=$user.my_library
sc/expand_deck deck=new_deck compile=fortran_input
sc/quit write_library=false
/fortran input=fortran_input
```

The USE_LIBRARY subcommand selects the library on file $USER.MY_LIBRARY as the base library. The EXPAND_DECK subcommand expands the deck named NEW_DECK. SCU reads the deck from the base library file and writes the expanded text on the file FORTRAN_INPUT. Because no changes were made to the library, the QUIT subcommand ends the SCU session and specifies that a result library is not written. The FORTRAN command calls the FORTRAN compiler to compile the text on the FORTRAN_INPUT file. (The FORTRAN command could have been entered within the SCU session.)

Selecting Decks

The first step in generating an expanded text file is selecting the decks to be expanded. You specify the decks to be expanded using the DECK parameter on the EXPAND_DECKS subcommand. You can also use selection criteria commands to select decks to be expanded if you specify the SELECTION_CRITERIA parameter on EXPAND_DECKS. If you do not specify the DECK parameter, the most recently used deck is expanded. To expand decks specified by selection criteria commands, use keyword NONE for the DECK parameter. Selection criteria commands are described later in this chapter.

The DECK parameter can specify a list of decks by name or by range. By default, the EXPAND_DECKS subcommand writes the decks on the expanded text file in the order they are listed in the working library deck list. To write the decks in the order you specify on the DECK parameter, specify ORDER=COMMAND on the EXPAND_DECKS subcommand.

For example, suppose the working library has the following deck list.

DECKA

DECKB

DECKC

DECKD

DECKE

The following EXPAND_DECKS subcommand selects four decks from the list.

```
sc/expand_decks deck=(deckd,decka..deckc) order=command
```

Because ORDER is specified as COMMAND, the EXPAND_DECKS subcommand writes the decks in the following order.

DECKD

DECKA

DECKB

DECKC

Copying Decks into a Compiler Input File

One of SCU's primary uses is to insert the text of one deck into the text of another deck before writing the deck to the expanded text file. This process is called copying decks.

For example, a deck to be copied into a compiler input file usually contains text used by more than one program. It could be text for a utility routine or a system routine stored on a source library available to many users. The CYBIL program interface procedure declaration text is stored on a system-supplied source library. To use a CYBIL program interface procedure, you must copy the procedure declaration text into your CYBIL program.

To copy a deck into another deck, add a COPY directive in the deck text where SCU is to insert the text of the other deck. The deck to be copied is named on the COPY directive.

SCU processes the COPY directive as it expands your deck. When it reads a COPY directive, it searches for the specified deck. Assuming it finds the deck, it expands the deck text and writes it on the expanded text file. SCU does not write the COPY directive to the expanded text file.

Text-Embedded Directives

The COPY directive is an SCU text-embedded directive. Chapter 4 describes the SCU text-embedded directives.

While expanding text, SCU recognizes a text-embedded directive when it reads a line beginning with the key character followed by a directive verb. The key character is the prefix character for all text-embedded directives in the library. The key character can be specified on the CREATE_LIBRARY subcommand. The default key character is *.

When it is looking for a deck that is specified in a COPY directive, SCU first searches in the working library and then examines the alternate base library decks.

For example, suppose both DECK1 and DECK2 reside on the source library file $USER.MY_LIBRARY and the key character for the source library is *. DECK1 contains the following text.

```
      program example
*copy deck2
      stop
      end
```

DECK2 contains the following text.

```
      do 10 i=1,100
10    i = i+1
```

Suppose you enter the following subcommand within an SCU session.

```
sc/expand_deck deck=deck1 compile=fortran_input
```

The subcommand expands the DECK1 text on the file FORTRAN_INPUT. While expanding the text, it processes the *COPY directive, copying DECK2 into DECK1. The following is the expanded text on the FORTRAN_INPUT file.

```
      program example
      do 10 i=1,100
10    i = i+1
      stop
      end
```

Copying Decks from Another Source Library

You can copy decks that reside on another source library into your compiler input file. The other source library is called an alternate base library. You use an alternate base library by specifying the name of the source library file that contains the decks to be merged on the ALTERNATE_BASE parameter of the EXPAND_DECK subcommand. SCU includes the alternate base library decks in the working library for the duration of the subcommand.

When it is looking for a deck that is specified in a COPY directive, SCU first searches the working library and then examines the alternate base library decks. For example, the following subcommand expands a CYBIL program that uses CYBIL program interface calls.

```
sc/expand_deck deck=deck10 compile=cybil_input ..
sc../alternate_base=$system.cybil.osf$program_interface
```

The subcommand expands DECK10 and writes the compiler input file on file CYBIL_INPUT. SCU copies program interface procedure decks from the source library file $SYSTEM.CYBIL.OSF$PROGRAM_INTERFACE as specified by COPY directives in the deck text.

Selecting Decks Using the Expand Attribute

A deck could contain text that is only to be inserted into another deck; the text is never expanded as a program in itself. For example, the deck could contain a set of CYBIL TYPE declarations, but not contain any executable statements. This type of deck is often called a common deck. If you expand common decks, use COPY or COPYC directives, not EXPAND_DECK subcommands. The EXPAND parameter on the CREATE_DECK subcommand determines whether an EXPAND_DECK subcommand expands the deck. The EXPAND parameter sets the expand attribute for the deck, which is stored in the deck header. By specifying that the expand attribute is FALSE when you create a deck, the deck can be expanded only when copied by a COPY or COPYC directive. Therefore, the expand attribute for common decks should be FALSE.

The expand attribute is useful when a library contains both types of decks. In this case, an EXPAND_DECK subcommand that specifies a range of decks or all decks within the library expands only those decks whose expand attribute is TRUE; it does not expand the common decks whose expand attribute is FALSE.

For example, suppose the following is the deck list and expand attributes for the working library.

Deck	Expand Attributes
COMMON1	FALSE
DECK1	TRUE
DECK2	TRUE

If you specify the EXPAND parameter on the CREATE_DECK subcommand, the deck will be expanded.

The following subcommand expands all decks whose expand attribute is TRUE.

```
sc/expand_deck deck=all
```

The subcommand writes the expanded text from DECK1 and DECK2 on the default compile file COMPILE. It skips deck COMMON1 and generates a warning message because its expand attribute is FALSE.

Generating an Expanded Text File from a File

In much the same way that you generate an expanded text file from a deck, you can also generate an expanded text file from a source file which you can then use as an input file to a compiler. When SCU expands a file, it processes the directives embedded in the source text and copies the expanded text to a separate SCU compile file. The file is expanded as though it were a deck on an SCU library.

You can expand text files during an SCU session by entering an EXPAND_FILE subcommand, or outside an SCU session by entering the NOS/VE command EXPAND_SOURCE_FILE.

The following example shows how to create and use a compiler input file within an SCU session.

```
/scu
sc/use_library base=$user.my_library
sc/expand_file file=new_file compile=fortran_input
sc/quit write_library=false
/fortran input=fortran_input
```

The USE_LIBRARY subcommand selects the library on file $USER.MY_LIBRARY as the base library. The EXPAND_FILE subcommand expands the file named NEW_FILE. SCU writes the expanded text on file FORTRAN_INPUT. The QUIT subcommand ends the SCU session, specifying that a result library is not written. The FORTRAN command calls the FORTRAN compiler to compile the text on the FORTRAN_INPUT file. (The FORTRAN command could have been entered within the SCU session.)

The next example uses the NOS/VE command EXPAND_SOURCE_FILE to produce the same result as the previous example without directly using SCU.

```
/expand_source_file file=new_file compile=fortran_input ..
../alternate_base=$user.my_library
/fortran input=fortran_input
```

Both the EXPAND_FILE subcommand and the EXPAND_SOURCE_FILE command process text-embedded directives within files as though they were within decks. Chapter 4 describes the SCU text-embedded directives.

To copy a deck into a compiler input file, for example, add a COPY directive in the file text where SCU is to insert the text of the specified deck. The deck to be copied is named on the COPY directive.

To copy decks from an alternate base library into your compiler input file, specify the ALTERNATE_BASE parameter on an EXPAND_SOURCE_FILE command; if you are in an SCU session, specify the ALTERNATE_BASE parameter on the EXPAND_FILE subcommand. The ALTERNATE_BASE parameter specifies the source library file containing the decks to be merged. SCU includes the alternate base library decks in the working library for the duration of the command or subcommand.

SCU processes the COPY directive as it expands your file. When it reads a COPY directive, it searches libraries for the specified deck. Assuming it finds the deck, it expands the text and writes it on the expanded file. SCU does not write the COPY directive to the expanded text file.

The following EXPAND_FILE subcommand expands a CYBIL program that uses CYBIL program interface calls.

```
sc/expand_file file=file_10 compile=cybil_input ..
sc../alternate_base=$system.cybil.osf$program_interface
```

Entering the subcommand expands FILE_10 and writes the compiler input file on file CYBIL_INPUT. When it is processing any COPY directives on FILE_10, SCU first searches for the specified deck on the working library and then searches the alternate base library. Thus, SCU copies any program interface procedure decks from the source library file $SYSTEM.CYBIL.OSF$PROGRAM_INTERFACE.

Editing Decks

Chapter 3 lists the EDIT_FILE subcommands that pertain to editing decks. These and all other subcommand descriptions for the EDIT_FILE utility are described in the NOS/VE File Editor manual.

Extracting Unexpanded Text

You can extract unexpanded text from a source library without processing directives embedded in the text. You could then use the unexpanded text as listing text, source text for a new deck, or input to another text processor.

The following subcommands extract unexpanded text.

- The EDIT_FILE subcommand WRITE_FILE copies lines from the deck being edited to the selected file.
- The SCU EXTRACT_DECK subcommand copies all lines from one or more decks to a source file.

Refer to the NOS/VE File Editor manual for a description of the WRITE_FILE subcommand.

To specify the decks an EXTRACT_DECK subcommand copies, specify the decks using the DECK parameter, selection criteria commands, or both.

Creating a New Library from an Existing Library

To create a source library that contains copies of decks from an existing source library, specify the existing source library as your base library (assuming you have read permission to the file). Using an existing source library as your base library does not affect the existing library.

Initially, your working library is a duplicate of the base library. However, during the SCU session, you can change both the library header information and the library deck list.

Changing the Library Header

To display the contents of the library header, enter a DISPLAY_LIBRARY subcommand. It displays the following header information.

- Library name
- Library version
- Version of SCU which created the library
- Source library format version
- Change counter
- Library description
- Creation date and time
- Last modification date and time
- Key character
- Most recently used deck and modification
- Number of decks, modifications, groups, and features in the library

Enter a CHANGE_LIBRARY subcommand to change the name, version, or description of the working library.

Changing the Library Deck List

The deck list is the list of decks copied to the new source library; it is ordered alphabetically by deck name. Initially, the working library deck list is a duplicate of the base library deck list. To view the deck list, enter the DISPLAY_DECK_LIST subcommand.

You can edit the deck list by deleting decks, creating decks, or merging decks from other source libraries into the deck list. (The process of merging libraries is described later.) You can also change the deck header information in existing decks.

Deleting Decks

To delete decks from the deck list, enter DELETE_DECKS and specify the decks to be deleted. You can specify them as a list, a range, or a list of ranges within the deck list.

Specify a range by naming the first and last deck in the range separated by an ellipsis (..). For example, the range DECKA..DECKC deletes all decks from DECKA through DECKC within the deck list. If a DECKB exists within the deck list, SCU deletes it because it is within the range.

Changing the Deck Header

To change deck header information, enter a CHANGE_DECK subcommand. It can change the following deck header items:

- Deck author
- Deck content description
- Processor to which the deck text is input after it is expanded (for example, CYBIL)
- Default tab character used in line mode only (if you do not define a tab character, line mode tabbing is disabled)
- Default tab columns
- Default line width when the deck is expanded
- Default line identifier placement
- Expand attribute

Entering a CHANGE_DECK subcommand also allows you to change the groups to which a deck belongs.

Deck Creation Options

The minimum requirements to create a deck have already been described. SCU provides the following additional capabilities for the CREATE_DECK subcommand.

- Copying information from another deck header for the header of the new deck
- Creating a multipartitioned deck
- Creating several decks with one subcommand

Copying a Deck Header

When creating a new deck, you can copy deck header information from a deck already in the working library. To display the header of an existing deck, enter DISPLAY_DECK and specify the deck name.

You can copy the entire deck header or only portions of it for the new deck. To do so, specify the name of the existing deck on the SAME_AS parameter of the CREATE_DECK subcommand that creates the new deck. If you do not want certain items copied, specify those items explicitly with CREATE_DECK parameters.

For example, the following subcommand creates a new deck containing the text from file TEXT_FILE.

```
sc/create_deck ..
sc../modification=firstmod ..
sc../deck=deck5 ..
sc../source=text_file ..
sc../same_as=deck4 ..
sc../author='Jane Doe'
```

The deck header of the new deck is identical to the deck header of DECK4, except for the deck name and author.

Creating a Multipartitioned Deck

When you are creating a deck, by default only one partition of data is copied to the new deck. However, if you specify MULTI_PARTITION=TRUE on the CREATE_DECK subcommand, deck text can consist of more than one partition.

If the MULTI_PARTITION parameter is FALSE, SCU copies text until the first end-of-partition is encountered. If the MULTI_PARTITION parameter is TRUE, SCU copies all text on the file, converting each end-of-partition to a WRITE_END_OF_PARTITION (WEOP) directive within the deck. When the deck is later expanded, the WEOP directives become end-of-partition delimiters.

Creating Multiple Decks

Entering a CREATE_DECK subcommand can also create more than one deck. The new deck headers are identical except for the deck names and, optionally, the expand attribute. You can specify an expand attribute for each deck name if you specify the deck names on DECK directives within the source text.

The way in which multiple decks are created differs, depending on whether you specify the deck names on the subcommand or in the source text file.

Deck Names Specified on the Subcommand

If you specify deck names on the CREATE_DECK subcommand, SCU copies text to the decks from the source files specified on the SOURCE parameter. The text for the first deck listed is copied from the first file listed, text for the second deck is copied from the second file listed, and so forth.

If you specify the SOURCE parameter, the number of source files specified must match the number of deck names specified. However, you can create empty decks by specifying the $NULL file as the source for the deck. For example, the following is the source file list for a subcommand that creates four decks, with the second and third decks empty.

```
SOURCE=(TEXT1,$NULL,$NULL,TEXT4)
```

If the MULTI_PARTITION parameter is FALSE, only one partition is copied; if the MULTI_PARTITION parameter is TRUE, all text on the file is copied.

Deck Names Specified in the Text

If you omit the DECK parameter and specify DECK_DIRECTIVES_INCLUDED=TRUE on the CREATE_DECKS subcommand, SCU creates a deck header for each DECK directive it finds in the source text file. (Only one source text file is read.)

The text following the DECK directive is copied to the deck. If the MULTI_PARTITION parameter is FALSE, it copies only the first partition. If the MULTI_PARTITION parameter is TRUE, it copies text until it encounters the end of the file and substitutes text-embedded directives (*WEOP) for the partition separator.

For example, suppose the source file SOURCE1 contains the following text.

```
*deck deck1
      program example
*copy decka
      stop
      end
*deck decka expand=false
      do 10 i=1,100
   10 i=i+1
```

The following subcommand would create two decks, DECK1 and DECKA.

```
sc/create_decks modification=firstmod source=source1 ..
sc../deck_directives_included=true
```

The CREATE_DECKS subcommand processes only the DECK directives; it does not process the COPY directive.

Suppose an EXPAND_DECKS subcommand specified both decks. Because EXPAND=FALSE was specified on the DECKA directive, the EXPAND_DECKS subcommand expands DECK1, but not DECKA. However, when it processes the COPY directive in DECK1, it expands the DECKA text and inserts it into the DECK1 text.

Conditional Text Expansion

When writing source text, you may require that the expanded source text be expanded differently. The text could differ by the content of a single line or by the inclusion or omission of a block of lines. Text can be expanded using string insertion or block expansion, as described in the following paragraphs.

String Insertion

You can change the content of a single line of expanded text with the PUT directive. When SCU processes a PUT directive during deck expansion, it evaluates the expression in the line before inserting the line in the expanded text.

The expression could use an SCL or SCU function. The NOS/VE System Usage manual describes the available SCL functions. Chapter 6 describes the available SCU functions.

For example, the following PUT directive concatenates strings to the string returned by the SCL function $DATE.

```
*put 'current_date = ''''//$date(month)//''''';'
```

Assuming the current date is May 3, 1985, SCU inserts the following line when it expands the text.

```
current_date = 'May 3, 1985';
```

Block Expansion

You can embed directives within source text so that you can include a block of lines when the text is expanded.

By default, the entire deck is a block. However, you can subdivide the deck into blocks using the following text-embedded directives.

- The BLOCK and BLOCKEND directives unconditionally delimit a block. The block contains conditional directives such as COPYC, which copies a deck only if the deck has not already been copied within its block.
- The TEXT and TEXTEND directives delimit a block of lines that are processed as text, not as text-embedded directives.
- The IF and IFEND directives delimit a block that is expanded only if the boolean expression on the IF directive is TRUE.

  For example, consider the following lines.

```
*block
*if (optimize=true)
*copy decka
*ifend
*copy deckb
*copyc decka
*blockend
```

  If the OPTIMIZE variable is TRUE:

  - The first COPY directive is processed and
  - SCU writes DECKA and then DECKB.

  The COPYC directive is not processed. If OPTIMIZE is FALSE:

  - The first COPY directive is skipped.
  - SCU writes DECKB, and then writes DECKA as a result of the COPYC directive.

- The ELSEIF and ELSE directives subdivide an IF/IFEND block. More than one ELSEIF directive can appear within an IF/IFEND block. An ELSEIF directive specifies a condition that is evaluated only if all previous statements in the IF block are FALSE.

  An ELSE directive can appear only once within an IF/IFEND block; it must follow all ELSEIF directives in the block. It does not specify a condition; the statements between ELSE and IFEND are expanded only if all conditions are evaluated as FALSE.

  For example, consider the following lines.

```
*if color='red'
*put 'You are dynamic'
*elseif color='white'
*put 'You are pure'
*elseif color='blue'
*put 'You are loyal'
*else
*put 'Try again'
*ifend
```

  The IF/IFEND block inserts a line of text, depending on the value of the COLOR variable.

Nesting Levels

Except for TEXT/TEXTEND blocks, you can nest blocks. A nested block is entirely within another block.

You can also nest directives. The lines inserted by a COPY or COPYC directive can contain additional directives. The DISPLAY_DECK_REFERENCES subcommand lists all decks that a specified deck copies and all decks that copy a specified deck. It can list both direct and indirect references.

An example of an indirect reference is if DECKA copies DECKB, which copies DECKC. In this case, DECKA indirectly references DECKC. A directive within DECKC would be in the third nesting level from DECKA.

```
*deck decka
    program example
*copy deckb
    stop
    end
*deck deckb expand=false
    integer i
*copy deckc
*deck deckc expand=false
    integer j
```

By default, SCU processes all nested levels of directives, although the sequence cannot be recursive. A recursive sequence is one in which a deck copies itself, either directly or indirectly.

You can limit the number of nested levels that are processed by using the EXPANSION_DEPTH parameter on the EXPAND_DECK subcommand. If, during expansion, SCU reads a directive at a level lower than the maximum expansion depth, it does not process the directive, but instead SCU leaves it in the expanded text.

Condition Specification

You specify a condition on an IF or ELSEIF directive as a boolean expression. Evaluation of the condition allows the expanded text to change without changing the source text. The condition must depend on an external condition, such as the value of a NOS/VE variable or the value returned by a NOS/VE or SCU function.

The NOS/VE System Usage manual describes the available NOS/VE functions and variables. Chapter 6 describes the available SCU functions.

Selecting Decks Using Selection Criteria Subcommands

Selection criteria processing allows you to specify which decks to extract or expand from a set of decks. The subcommands you use to select the decks are called selection criteria subcommands. You can store selection criteria subcommands in a separate text file or enter them directly at the terminal.

To use selection criteria subcommands located in a separate text file, specify the file on the SELECTION_CRITERIA parameter of the

- EXPAND_DECK, EXTRACT_DECK, or EXPAND_FILE subcommand or
- EXTRACT_SOURCE_LIBRARY or EXPAND_SOURCE_FILE command.

SCU reads commands from the file until it reads the QUIT subcommand or reaches the end of the file.

To enter selection criteria subcommands from an interactive terminal, specify COMMAND on the SELECTION_CRITERIA parameter. SCU asks you for commands with the scc/ prompt and continues to prompt until you enter the QUIT subcommand.

Chapter 5 describes the selection criteria subcommands. These subcommands explicitly include or exclude parts of a library.

Using selection criteria subcommands, you can include or exclude decks by name or by the group to which they belong. You can include or exclude decks from the base library or an alternate base library.

Similarly, you can exclude modifications by name or by the feature with which they are associated. When you exclude a modification, the modification changes are not included in the text of the file.

For example, the following subcommand shows you how to exclude all changes belonging to modification MOD1.

```
scc/exclude_modification mod1
```

Expanding Decks that Reference a Common Deck

Making a change to a common deck often requires that you recompile each deck that references the common deck. To get an expanded text file containing all decks that reference the common deck, enter the selection criteria subcommand INCLUDE_COPYING_DECKS.

For example, the following EXPAND_DECK subcommand expands all decks in the working library that copy the deck COMMON_DECK.

```
sc/expand_deck decks=all selection_criteria=command
scc/include_copying_decks deck=common_deck
scc/quit
sc/
```

Excluding a Common Deck Library

You can copy common decks from an alternate base library without expanding the alternate base library decks. You do so by excluding the alternate base library with the selection criteria subcommand EXCLUDE_LIBRARY.

For example, the following subcommand expands all decks in the working library, but it does not expand decks on the alternate base library COMMON_LIBRARY.

```
sc/expand_decks decks=all alternate_base=common_library ..
sc../selection_criteria=command
scc/exclude_library common_library
scc/quit
sc/
```

The EXPAND_DECKS subcommand expands all decks in the working library whose expand attribute is TRUE. When searching for a deck specified on a COPY or COPYC directive, EXPAND_DECKS searches the alternate base library if the deck is not in the working library.

Entering Other Control Statements

Besides selection criteria subcommands, you can enter SCU display subcommands and SCL control statements in response to an scc/ prompt. The scc/ prompt appears whenever selection criteria subcommands are being entered.

SCL control statements within the criteria file can create and assign values to SCL variables. An expression on an IF, ELSEIF, or PUT text-embedded directive can reference an SCL variable assigned in the selection criteria file or elsewhere.

For example, suppose you enter the following SCL statement in response to a selection criteria prompt.

```
scc/optimize = true
```

The statement specifies that the OPTIMIZE variable is TRUE. Either of the following IF directives in the source text reference the SCL variable.

```
*if (optimize = true)
      or
*if optimize
```

Because the expression is TRUE, SCU expands the subsequent block of lines.

Editing the Modification List

Editing modifications is a way of organizing and recording changes you make to a source library. By associating a set of changes with a modification, you can save and restore that set of changes. You can list the modifications in a library by entering the DISPLAY_MODIFICATION_LIST subcommand.

You can also change the level of authority associated with a modification to allow or deny access to lines in any deck that belong to the modification.

Creating a Modification

You can create a modification in the following ways:

- Use the CREATE_DECK subcommand to create a deck.
- Use the CREATE_MODIFICATION subcommand to predefine a modification.
- Specify the new modification as the MODIFICATION parameter on the EDIT_DECK subcommand that begins an editing session. All changes that you make to any deck within an editing session are associated with that modification name.

By using a CREATE_MODIFICATION subcommand to predefine a modification, you can specify modification header information when you create the modification. You can also specify modification header information with a CHANGE_MODIFICATION subcommand.

The modification header describes the modification. It contains the following items.

- Modification name
- Feature to which the modification belongs
- Modification author
- Modification description

To list the current contents of a modification header, enter a DISPLAY_MODIFICATION subcommand.

Deleting a Modification

To remove all text changes associated with a modification, enter a DELETE_MODIFICATION subcommand. After you delete a modification, all changes introduced by the modification are reversed. All insertions are deleted, all replacements are removed, and all deletions are reactivated.

When a line is deleted, it becomes an inactive line. SCU does not expand inactive lines when it expands the deck. When a line is reactivated, it is once again an active line and is included in a deck expansion.

Changing Your Modification States and Authority

During a project, source text can pass through several phases of development. SCU associates a development state with each modification. It recognizes the following states.

State	Development Phase
0	Experimental (text and header changes allowed)
1	Developmental (header changes only)
2	Stable (header changes only)
3	Verified (header changes only)
4	Released (no changes allowed)

In states 1, 2, and 3, changes can be made to the modification header, but not to modification text.

NOTE

Modifications in state 4 cannot be lowered in state, and their header and text information cannot be changed. Changes to lines introduced in state 4 must be made under a new modification.

The initial state of a modification is 0. You can display the current state of a modification by entering the DISPLAY_MODIFICATION subcommand.

SCU only allows users with the appropriate authority to change the state of a modification. You can raise a modification state up to your authority for the file. For example, if your authority for a file is 3, you can raise the state to 1, 2, or 3, but not to 4.

Your authority for a file is 0, unless you are granted higher authority by the NOS/VE command CREATE_FILE_PERMIT. Your authority is indicated by a digit (1 through 4) in the application information field of your file permit entry. Interlocking decks in a library are allowed if you include the character I in the application information field.

Examples:

To assign read-only access to a file, enter:

```
/create_file_permit file=$user.fname group=public ..
../access_mode=read share_mode=none
```

To assign yourself the highest authority (unrestricted access) to a source library file, enter:

```
/create_file_permit file=$user.fname group=user ..
../access_mode=(all cycle control) share_mode=none ..
../application_information='I4'
```

To assign access authority so that the modification status can be changed later, enter:

```
/create_file_permit file=$user.fname group=public ..
../access_mode=(read, modify) share_mode=none ..
../application_information='4'
```

To display your authority for a file, enter the following NOS/VE command to display the application_information string in your file permit entry.

```
/display_catalog_entry file=$user.fname ..
../display_option=permits
```

The CREATE_FILE_PERMIT and DISPLAY_CATALOG_ENTRY commands are described in the NOS/VE System Usage manual.

Assuming you have the required authority, you can change a modification state by entering a CHANGE_MODIFICATION subcommand, which raises the state or lowers it to 0. Lowering the state to 0 allows you to change the modification text.

You can lower the state of a modification to 0 only if your authority is greater than or equal to the current state of the modification. For example, if the modification state is 2, you can lower it only if your authority is 2, 3, or 4.

Modification states and authority also affect the DELETE_DECK and DELETE_MODIFICATION subcommands as follows.

- You can delete only those decks whose creation modification state is no greater than your authority.
- You can delete only those modifications whose state is no greater than your authority.

If you specify a deck or modification that you cannot delete because you lack authority, SCU skips that deck or modification and continues processing with the next deck or modification specified on the command.

NOTE

The record of deck header changes cannot be reversed. If an attribute such as EXPAND is changed along with text changes, it is not reversed if a MODIFICATION is removed. Only the text changes that belong to a MODIFICATION are easily removed.

Modification states and authority also limit use of the SEQUENCE_DECK subcommand and EXTRACT_SOURCE_LIBRARY command as described later in this chapter.

Sequencing Line Identifiers

Sequencing assigns new line identifiers to each line of a modification or deck.

Sequencing a Modification

You can sequence a modification using the SEQUENCE_MODIFICATION subcommand, which adjusts the sequence numbers to top-down order.

During an editing session, SCU assigns sequence numbers in the same order in which the lines are introduced in the modification. The sequence numbers remain in the order in which they were introduced.

For example, suppose you entered the EDIT_DECK subcommand specifying MY_MOD as the modification name. The editing session includes the following subcommand.

```
sce/insert_lines
Enter text
?       do 10 i=1,10
?    10 n(i)=n(i-1) + i
?**
```

SCU assigns the two lines that were inserted the line identifiers MY_MOD.1 and MY_MOD.2.

```
my_mod          1       do 10 i=1,10
my_mod          2    10 n(i)=n(i-1)+i
```

Suppose the next editing session also specified the modification name MY_MOD and contained the following subcommands.

```
sce/replace_text text='i=1,10' new_text='i=1,20'
sce/insert_line new_text='n(i)=1' placement=before
```

SCU gives the changed line the line identifier MY_MOD.3 and the inserted line the identifier MY_MOD.4. After you exit the editing session, the deck appears in the following order:

```
my_mod          4       n(i)=1
my_mod          3       do 10 i=1,20
my_mod          2    10 n(i)=n(i-1) + i
```

Sequencing a Deck

You can sequence a deck when the creation modification for the deck has been raised to released state (state 4), and you have this level of access for the file. Sequencing a deck renumbers the line identifiers of all lines in the deck that belong to modifications in state 4.

Sequencing a deck also discards all lines deleted by modifications in state 4. New line identifiers are assigned, consisting of the specified modification name and a sequence number. Sequence numbers are assigned in the order the lines appear in the deck. After they have been sequenced, all deck lines in state 4 belong to the modification specified in the SEQUENCE_DECK command.

NOTE

Line identifiers for deck lines belonging to modifications not in state 4 are not changed.

Enter the SEQUENCE_DECK subcommand to sequence a deck. You can sequence the deck whenever a modification in the deck is raised to released state. Changes cannot be made to a modification that is in released state. Sequencing a deck consolidates the modifications that cannot be changed, thus minimizing the number of modifications in the deck and freeing additional file space. Lines in a deck which were introduced by modifications at states less than 4 maintain their original identity.

Extracting a Source Library

You can extract a source library by creating a new library containing one or more decks copied from an existing source library.

Extracting a library allows you to edit copies of a subset of the decks on an existing library for future use. Later, the edited version of the extracted library can be merged with the old library to create a new library. The user who merges the libraries must have read access permission to your extracted library and to the original library; the user must also have modify, shorten, and append access permission to the new library file.

To extract a library, enter the command EXTRACT_SOURCE_LIBRARY before starting your SCU session. The BASE parameter specifies the file containing the existing source library and the RESULT library specifies the file on which the extracted library is written. Then, when you enter the SOURCE_CODE_UTILITY command to begin your session, specify the extracted library file on the BASE parameter of the USE_LIBRARY subcommand.

For example, the following commands show how to extract a source library and begin an SCU session using the extracted source library.

```
/extract_source_library decks=(deck114..deck120) ..
../base=$system.library result=$user.my_library ..
../interlock=none
/scu
sc/use_library base=$user.my_library ..
sc../result=$user.my_library.$next
sc/
```

Using Interlocks

When you extract a set of decks from a library using the EXTRACT_SOURCE_LIBRARY command, you may want to interlock them so you can:

- Notify other users that you have extracted a copy of the deck and that any changes they make to the deck will not be included in the future version of the library.
- Prevent other users from interlocking the deck.

- Prevent other users from merging a deck list containing a copy of the interlocked deck with the original deck list to create a new library.

Setting an interlock sets the interlock fields. Two interlock fields exist in a deck header: an original interlock field and a subinterlock field. When you interlock decks, the subinterlock field is set in the deck header on the base library and the original interlock field is set in the deck header on the extracted library.

Setting an Interlock

To interlock decks, you must specify the INTERLOCK parameter on the EXTRACT_SOURCE_LIBRARY command that extracts the deck. The value specified on the INTERLOCK parameter is stored in the appropriate interlock field in the deck header, with the date and time the interlock was performed.

To interlock decks in a library, you must have modify and read permission to the library file and the character I (for interlock) specified in the application_information field of your file permit entry. Permissions and the application_information string are specified on a CREATE_FILE_PERMIT command as described in the NOS/VE System Usage manual.

For example, to assign access authority allowing decks to be interlocked on the library, enter:

```
/create_file_permit file=$user.fname group=public ..
../access_mode=(read,modify) share_mode=none ..
../application_information='I'
```

An interlocked deck must be extracted whole. Modifications to the deck cannot be excluded by selection criteria directives.

Nesting Interlocks

You can nest interlocks: in other words, you can extract an interlocked library from another interlocked library.

Figure 1-4 shows an example of nested interlocks. In the example, user 1 extracts library B from library A, setting interlocks on decks X and Y. The subinterlock field is set for the interlocked library A decks (X and Y) and the original interlock field for the library B decks. Then, user 2 extracts library C from library B, setting an interlock on deck Y. The subinterlock field is set for the interlocked library B deck and the original interlock field for the library C deck. Therefore, the copy of deck Y on library B has both its original interlock and subinterlock fields set.

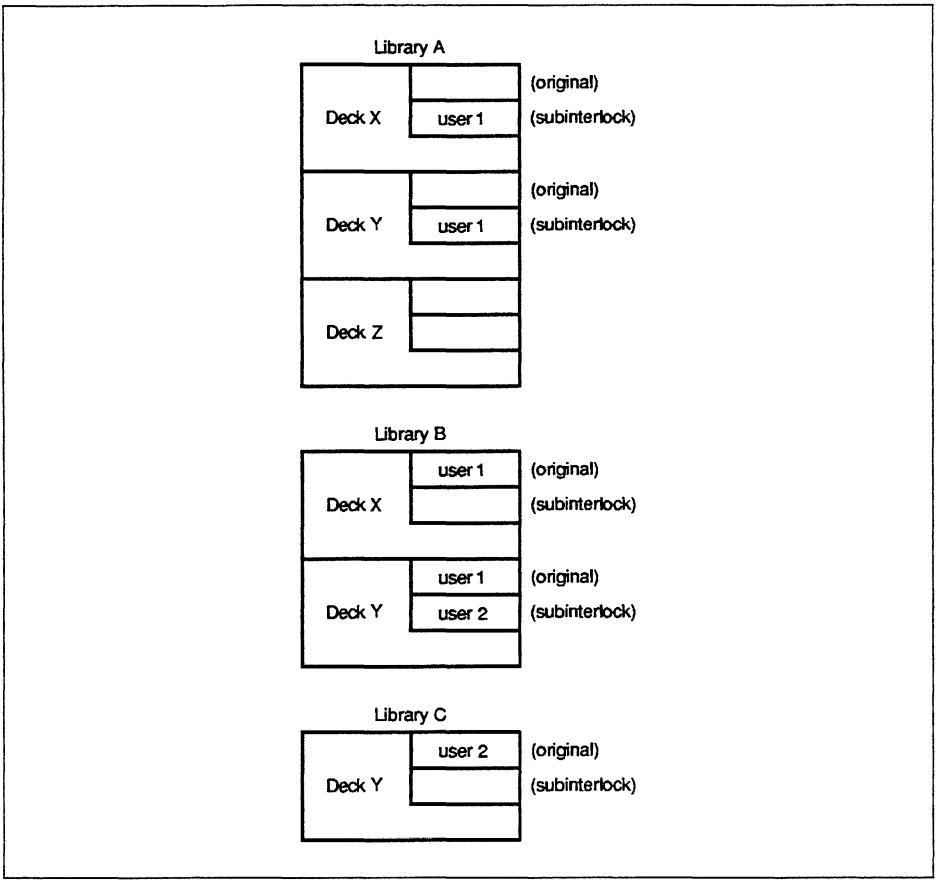


Figure 1-4. Example of Nested Interlocks

Restrictions of Interlocking

Interlocking a deck restricts its use as follows:

- If you enter the EXTRACT_SOURCE_LIBRARY command to set an interlock on a deck whose subinterlock is already set:
  - A warning message appears.
  - The interlocked deck is skipped.
  - Remaining decks are processed.
- If you enter a COMBINE_LIBRARY or REPLACE_LIBRARY subcommand to merge the extracted library with the base library:
  - A warning message appears.
  - Nonmatching interlocked decks are skipped.
  - Merging of matching decks continues.

  Decks are nonmatching if the following two conditions apply:
  - The ENFORCE_INTERLOCKS parameter is set to TRUE.
  - The original interlock field of the extracted library decks does not match the subinterlock field of the interlocked base library decks.
- If you edit or sequence an interlocked deck, a warning message appears. The message lets you know that changes will be omitted from any new library created by merging the extracted decks with the decks on the existing library.

Clearing an Interlock

When you merge extracted decks with the decks in the original library to form a new library, SCU checks the contents of the interlock fields only if you request interlock enforcement. If requested by the ENFORCE_INTERLOCKS parameter on a COMBINE_LIBRARY or REPLACE_LIBRARY subcommand, SCU checks that the subinterlock field of the base deck header matches the original interlock field of the extracted deck header. If the fields match, the base deck is replaced by the extracted deck and the interlock fields are cleared on the result library.

For example, suppose you merge the libraries shown in figure 1-4 to form library D. Assuming interlocks are enforced, the library C copy of deck Y can only replace the library B copy of deck Y. It cannot replace the library A copy, because its original interlock field does not match the subinterlock field of the library A copy. You cannot merge library C with library A to form library D if interlocks are enforced. You could merge library C with library B, or library B with library A, or you could merge all three libraries to form a new library (described later in this chapter).

SCU provides a way of clearing the subinterlock field without merging libraries. A user with access authority 4 for the library file can clear a subinterlock field in a deck header by entering a CHANGE_DECK subcommand.

Merging Libraries

You can merge decks into the working library by entering ADD_LIBRARY, REPLACE_LIBRARY, and COMBINE_LIBRARY subcommands. The functions of each subcommand is as follows:

Subcommand	Function
ADD_LIBRARY	Adds new decks.
REPLACE_LIBRARY	Replaces existing decks.
COMBINE_LIBRARY	Adds new decks and replaces existing decks.

Each subcommand specifies one or more source libraries whose decks can be merged into the working library. If all the decks are new, enter the ADD_LIBRARY subcommand. If all the decks are copies of existing decks extracted from the base library, enter the REPLACE_LIBRARY subcommand. If the decks include both new decks and copies of existing decks, enter the COMBINE_LIBRARY subcommand. These subcommands do not change the contents of the working library header.

The following paragraphs describe how each subcommand selects the decks that are copied to the working library.

Adding Libraries

The ADD_LIBRARY subcommand compares the decks in each source library with the decks already in the working library. It compares each source library in the order you specify the libraries on the subcommand.

For each deck name it reads, SCU determines if a deck with that name already exists in the working library. If it does not, ADD_LIBRARY copies the new deck to the working library. If it is, ADD_LIBRARY does not copy the deck. If the deck is in both the source library and the working library, a warning message appears.

For example, suppose an ADD_LIBRARY subcommand specifies source libraries SOURCEA and SOURCEB. SCU first searches the deck list of SOURCEA. The following shows the SOURCEA deck list, the working library deck list, and the working library deck list after the SOURCEA search is complete. A warning message appears, describing the DECK1 duplication.

SOURCEA	Working Library	Intermediate Working Library
DECK1	DECK1	DECK1 (from working library)
DECK3	DECK2	DECK2 (from working library)
		DECK3 (from SOURCEA)

SCU then searches the deck list of SOURCEB. The following shows the SOURCEB deck list, the intermediate working library deck list, and the new working library after the SOURCEB search is complete. Because DECK3 is included in both SOURCEA and SOURCEB deck lists, it is not included in the working library deck list and a warning message appears, describing the DECK3 duplication.

SOURCEB	Intermediate Working Library	New Working Library
DECK3	DECK1 (working library)	DECK1 (working library)
DECK4	DECK2 (working library)	DECK2 (working library)
	DECK3 (SOURCEA)	DECK4 (SOURCEB)

Replacing Libraries

The REPLACE_LIBRARY subcommand compares the decks in each source library with the decks already in the working library. It compares each source library in the order you specify the libraries on the subcommand.

For each deck name it reads, SCU determines if the deck with that name already exists in the working library. If the deck is already in the working library, REPLACE_LIBRARY replaces the existing deck with the deck from the source library if it has not already been replaced. If the deck is not in the working library, REPLACE_LIBRARY does not copy the deck and a warning message appears.

For example, suppose a REPLACE_LIBRARY subcommand specifies source libraries SOURCEA and SOURCEB. SCU first searches the deck list of SOURCEA. The following shows the SOURCEA deck list, the working library deck list, and the working library deck list after the SOURCEA search is complete. A warning message appears, stating that DECK4 is not on the base library.

SOURCEA	Working Library	Intermediate Working Library
DECK1	DECK1	DECK1 (from SOURCEA)
DECK4	DECK2	DECK2 (from working library)
	DECK3	DECK3 (from working library)

SCU then searches the deck list of SOURCEB. The following shows the SOURCEB deck list, the intermediate working library deck list, and the new working library after the SOURCEB search is complete. DECK1 is not replaced by SOURCEB because it has already been replaced by SOURCEA.

SOURCEB	Intermediate Working Library	New Working Library
DECK1	DECK1 (SOURCEA)	DECK1 (SOURCEA)
DECK2	DECK2 (working library)	DECK2 (SOURCEB)
	DECK3 (working library)	DECK3 (working library)

Combining Libraries

The COMBINE_LIBRARY subcommand compares the decks in each source library with the decks already in the working library. It compares each source library in the order you specify the libraries on the subcommand.

For each deck name it reads, SCU determines if the deck with that name already exists in the working library. If the deck is already in the working library and has not been replaced by a previous source library copy, it replaces the existing deck with the deck from the source library. If it is not in the working library, it adds the deck from the source library.

For example, suppose a COMBINE_LIBRARY subcommand specifies source libraries SOURCEA and SOURCEB. SCU first searches the deck list of SOURCEA. The following shows the SOURCEA deck list, the working library deck list, and the working library deck list after the SOURCEA search is complete.

SOURCEA	Working Library	Intermediate Working Library
DECK1	DECK1	DECK1 (from SOURCEA)
DECK4	DECK2	DECK2 (from working library)
	DECK3	DECK3 (from working library)
		DECK4 (from SOURCEA)

SCU then searches the deck list of SOURCEB. The following shows the SOURCEB deck list, the intermediate working library deck list, and the new working library after the SOURCEB search is complete. DECK4 is not replaced because it has already been replaced by SOURCEA.

SOURCEB	Intermediate Working Library	New Working Library
DECK2	DECK1 (SOURCEA)	DECK1 (SOURCEA)
DECK4	DECK2 (working library)	DECK2 (SOURCEB)
	DECK3 (working library)	DECK3 (working library)
	DECK4 (SOURCEA)	DECK4 (SOURCEA)

Substituting Deck Names

SCU has two subcommands that substitute new names in a result library for old names in the base library: CHANGE_DECK_REFERENCES and CHANGE_DECK_NAME. For both subcommands, you specify the name substitutions on a separate file.

To change all deck references on COPY and COPYC directives, enter a CHANGE_DECK_REFERENCES subcommand. For a list of all references to the deck before changing the deck references, enter a DISPLAY_DECK_REFERENCES subcommand.

To change a deck name both within the deck header and on COPY and COPYC directives, use a CHANGE_DECK_NAME subcommand.

SCU reads the name substitutions from the file specified on the NAME_LIST parameter on the subcommand. You should specify one name substitution per line in the file. The substitution is specified as an SCL parameter list containing the following parameters.

OLD_NAME (ON)

Existing name.

NEW_NAME (NN)

Substituted name.

For example, any of the following lines changes DECKA to FIRST_DECK.

```
old_name=decka new_name=first_deck

on=decka, nn=first_deck

decka,first_deck
```

Generating Editor Subcommands

SCU provides two means of generating sequences of editor subcommands.

- The EXTRACT_MODIFICATION subcommand generates a sequence of editor subcommands that can reproduce the modification.
- The GENERATE_SCU_EDIT_COMMANDS command compares the text in a deck with that in a separate file and generates a sequence of editor subcommands that change the deck text to match file text. (GENERATE_SCU_EDIT_COMMANDS is a NOS/VE command that is described in the next chapter.)

NOTE

Changes made to the deck header such as changing the expand attribute to TRUE or FALSE) are not part of a modification set. Hence, these types of updates are not included in the GENERATE_SCU_EDIT_COMMANDS output.

The editor subcommand sequence generated by these two subcommands consists of the INSERT_LINES, REPLACE_LINES, and DELETE_LINES subcommands and the text they insert. Refer to the NOS/VE File Editor manual for descriptions of these subcommands.

Commands and Subcommands 2

Commands and Subcommands 2

The first part of this chapter describes SCL command use, abbreviations, parameters, and other syntax elements. It discusses initiating utilities in line or screen mode and grouping and referencing files. The second part of the chapter describes each SCU subcommand, each SCL command that initiates SCU execution, and other related SCL commands. The descriptions are presented in alphabetical order by name.

The following sections describe how to use NOS/VE commands and utilities and how to get information about commands from the system.

Using NOS/VE Commands

The following sections describe how to use NOS/VE commands and utilities and how to get information about commands from the system.

Names

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

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

Table 2-1 contains a list of characters you can use when forming names.

Table 2-1. Valid Characters for NOS/VE Names

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

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

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

Commands

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

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

- Commonly used verbs in commands are CREATE, DELETE, DISPLAY, CHANGE, and SET.
- Commonly used objects are FILE, CATALOG, and PROGRAM_ATTRIBUTES.

Since NOS/VE command names describe what they do, you can easily recognize the purpose of commands and anticipate the names for commands you have not yet used. For example, the DELETE_FILE command deletes files and the DISPLAY_FILE_ATTRIBUTES command displays file attributes.

Abbreviating Commands

You can abbreviate commands by using the first three letters of the verb followed by the first letter of each word in the object without underscores. For example, the following commands are shown with their abbreviations.

COPY_FILE or COPF

SET_WORKING_CATALOG or SETWC

The word PASSWORD when used in command or parameter names is abbreviated PW.

Continuing a Command on a New Line

To continue a command on another line, follow these steps:

1. End the current line with an ellipsis (..).
2. Press the return key. During an interactive session, the system prompts for continued lines by preceding the input prompt with an ellipsis. For example:

   ../

3. Continue the line.

The following example illustrates a command that is continued on a second line:

```
/copy_file ..
../input=infile output=..
../outfile
/
```

Using Parameters in Commands

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

parameter = value list

Parameter Names

Parameter names consist of one or more words, separated by underscores, and like command names, they describe what they do.

You must separate parameters from the command and from other parameters using spaces or commas, as the following examples show:

```
/copy_file input=infile output=outfile
/change_terminal_attributes,terminal_model=dec_vt220
```

You can abbreviate parameter names by specifying the first letter of each word in the parameter name. For example, you can abbreviate the preceding commands and parameters as follows:

```
/copf i=infile o=outfile
/chata tm=dec_vt220
```

The following are exceptions to this standard:

- STATUS has no abbreviation.
- PASSWORD is abbreviated PW.
- OUTPUT_DESTINATION is abbreviated ODE.
- OUTPUT_DISPOSITION is abbreviated ODI.
- Any parameter beginning with MAX or MIN (such as MAXIMUM_WORKING_SET and MINIMUM_WORKING_SET) preserves those three characters as part of its abbreviation (MAXWS and MINWS). In this way, the uniqueness of these parameters is preserved.

Parameter Order

You can also specify parameter values without specifying the parameter name, provided that you follow the order defined for the parameters. If you specify parameter values positionally, use a comma as a place holder for any unspecified parameters. The following example shows the EDIT_FILE command with values specified for the first and third parameters, and a comma used as a place holder for the second parameter.

```
/edit_file my_file,,output_file
```

Parameter Types

Every parameter is defined as a certain type (such as integer or boolean). The system verifies that the value you assign a parameter matches its defined type. If there is a mismatch, an error message is displayed.

The following table shows the different parameter types and some examples of the values you can assign.

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

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

A value list specifies one or more values to be processed for a parameter. To specify more than one value in a value list, enclose the list in parentheses and separate the values by a space or comma, as in the following example:

```
/detach_file file=(file1,file2)
```

Every NOS/VE command has an optional STATUS parameter which you can use to check for error conditions. If an abnormal condition occurs during execution of the command, the system passes information about that condition to the variable supplied for the STATUS parameter. You can include commands in your job to check the contents of the status variable and change the flow of execution based on the result.

Entering Commands

The two most common ways of executing commands are as follows:

- Enter the name of a command from the $SYSTEM command list or the name of a command on an object library you have added to the command list.
- Enter the name of an executable file.

For example, the following entry causes the DISPLAY_CATALOG command found in your command list to be executed:

```
/display_catalog
```

You execute the commands in a file by specifying the path of the executable file. The following entry causes the commands in $LOCAL.SAMPLE_PROC to be executed:

```
/$local.sample_proc
```

File paths are discussed later in this chapter.

Command Utilities

Command utilities are commands that make a set of *subcommands* available to you. The subcommands perform the operations of the utility. For example, the EDIT_FILE utility provides subcommands for editing files; the CREATE_OBJECT_LIBRARY utility provides subcommands for maintaining object libraries. Utility subcommands obey the same rules and conventions as other commands.

When you enter the command to initiate a utility, the system adds the subcommands associated with the utility to the list of NOS/VE commands already available. While the command utility is executing, you can enter both NOS/VE commands and the utility's subcommands. When you exit from a utility, however, its subcommands are removed from the command list and are no longer available for your use.

Initiating Utilities

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

```
/create_object_library
COL/
```

All utilities have one subcommand in common: QUIT. Entering this subcommand ends the utility.

Line and Screen Mode

Many utilities operate in both line mode and screen mode.

In line mode, you perform line-by-line operations by entering a subcommand (or other NOS/VE command) after the system prompt (/).

In screen mode, you interact with the utility by pressing keys associated with specific operations (such as moving or deleting lines or characters). The operations and their associated keys are displayed at the bottom of your screen.

Before you use a utility in screen mode, you must first identify the type of terminal you are using by entering the name of your terminal model on the TERMINAL_MODEL parameter of the CHANGE_TERMINAL_ATTRIBUTES command and then the command that changes your interactive style:

```
/change_terminal_attributes terminal_model=dec_vt220
/change_interaction_style style=screen
```

Because you typically identify your terminal once at the beginning of an interactive session, you could place the preceding commands, as well as those mentioned in the next section, in your user prolog (typically file $USER.PROLOG). All commands in your user prolog are processed by NOS/VE each time you log in.

Controlling Processing in Screen Mode

To be able to interrupt or terminate the processing of a utility operation or an SCL command while you are in screen mode, you must do the following:

1. Define an attention character.

   If you are logged in through CDCNET, you will enter this character to interrupt processing.

   If you are logged in through NAM/CCP, you will use control-**T** and control-**P** as you do in line mode to interrupt processing.

2. If you are a CDCNET user, specify how the network is to respond when you enter the attention character. This is called the attention character action. Specify a value of 1 if you want the network to interrupt operations, or 2 if you want the network to terminate operations.

The following sections give examples of establishing control processing through the NAM/CCP, NAM/CDCNET, and NAMVE/CDCNET networks. If you are not sure which network you use, ask your site personnel. The NOS/VE commands used in the examples are described in the NOS/VE System Usage manual. The CDCNET commands used are described in the CDCNET Terminal Interface Usage manual.

NAM/CCP Example

To be able to use control-T to terminate a command and control-P to interrupt operations while in a screen mode application, add the following NOS/VE command to your user prolog:

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

NAM/CDCNET Example

To use control-P to interrupt operations while in a screen mode application, add the following CDCNET commands to your user prolog:

```
%change_terminal_attributes attention_character=10(16)
%change_connection_attributes attention_character_action=1
```

NOTE

In line mode, the network accepts the character sequences %1 to interrupt processing and %2 to terminate processing. It also accepts control-P as the pause break sequence if you have entered the above commands in your user prolog.

NAMVE/CDCNET Example

To use control-T to terminate operations while in a screen mode utility, add the following NOS/VE commands to your user prolog:

```
change_terminal_attributes attention_character=20
change_term_conn_defaults attention_character_action=2
change_connection_attributes terminal_file_name=input aca=2
change_connection_attributes terminal_file_name=output aca=2
change_connection_attributes terminal_file_name=command aca=2
```

NOTE

In line mode, the network accepts the character sequences %1 to interrupt processing and %2 to terminate processing. It also accepts control-T as the terminate break sequence if you have entered the above commands in your user prolog.

NOS/VE Files

NOS/VE stores all data and programs in files. Files, in turn, are stored in logical groupings called catalogs. During an interactive job, you have access to a catalog of temporary files (called the $LOCAL catalog) and a catalog of permanent files (called the $USER catalog).

Temporary Files

Temporary files are created during an interactive job and are used as work files. When your interactive job ends, all temporary files are deleted.

The temporary files you create are stored in the $LOCAL catalog. In addition, NOS/VE uses the $LOCAL catalog to store certain system-defined files which it requires to process your job.

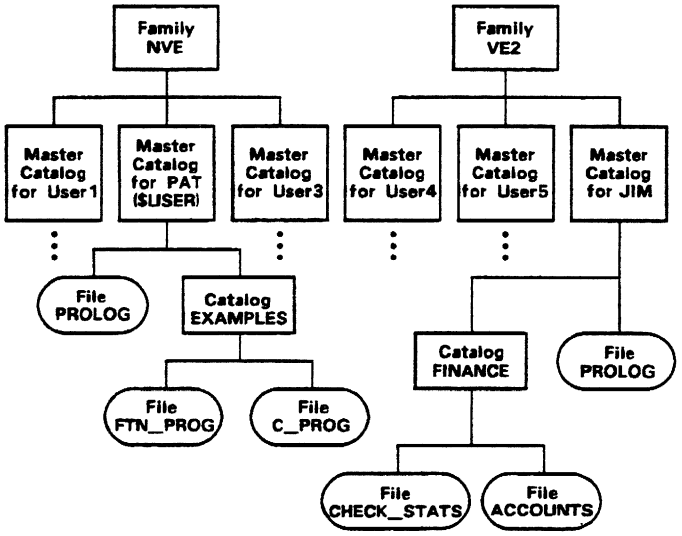
The $LOCAL catalog contains files only; you cannot create catalogs within $LOCAL.

Permanent Files

Permanent files are files NOS/VE saves in permanent catalogs under your user name. Permanent files are grouped into a hierarchy. From top to bottom, this hierarchy is described as follows:

Element	Description
Family	A grouping of NOS/VE users that determines the location of their permanent files. Each system can have more than one family. You can also refer to your family name as $FAMILY.
Master Catalog	Catalog that contains all the permanent files and subcatalogs belonging to a particular user. Users can refer to their master catalog by their user name or by the name $USER.
Subcatalog	Catalogs residing within the master catalog. Subcatalogs can contain both files and other subcatalogs.
File	A collection of information referenced by a name and residing in a user's master catalog or subcatalog.

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



Referencing Files

You reference a file by specifying its file path. A full file path includes the family name and all the catalogs in the path leading to the file. For example, the full file path for file FTN_PROG shown in the previous illustration is:

```
:NVE.PAT.EXAMPLES.FTN_PROG
```

You can reference files within your own family of users and within other families. Suppose you want to reference user JIM's CHECK_STATS file. File CHECK_STATS is in JIM's subcatalog FINANCE, and JIM's master catalog is part of family VE2. The correct file reference, then, is as follows:

```
:VE2.JIM.FINANCE.CHECK_STATS
```

Files can have more than one version called a file cycle. You can also specify a file position. Suppose the file FTN_PROG has three cycles and you want to specify that the third cycle be positioned at the end-of-information when it is first opened. The correct file reference, then, is as follows:

```
:NVE.PAT.EXAMPLES.FTN_PROG.3.$EOI
```

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

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

Using List Files Within SCU

You can establish a default listing file for the SCU subcommands by specifying the LIST parameter of the SET_LIST_OPTIONS subcommand. The default listing file is $LIST. You can override the default for the duration of a subcommand by specifying a different listing file on its LIST parameter.

For certain subcommands, you can select a brief or full listing with the DISPLAY_OPTIONS parameter. The parameter description indicates how the parameter changes the content of a listing.

You can also change the format of the listing by changing the attributes of the listing file (refer to the SET_FILE_ATTRIBUTE command description in the NOS/VE System Usage manual). If you do not specify the PAGE_WIDTH file attribute, the default is what was set for the terminal.

For listing on a terminal, you may have to reset the PAGE_WIDTH and PAGE_LENGTH terminal attributes to match the number of characters allowed on your screen (refer to the CHANGE_TERMINAL_ATTRIBUTE command description in the NOS/VE Commands and Functions manual).

The PAGE_FORMAT file attribute determines how often titles are inserted in a listing. The PAGE_WIDTH attribute determines the title format used.

SCU provides two formats of titles, one for 132-character lines, the other for 72-character lines. If the PAGE_WIDTH value is at least 132, the 132-column title format is used. Otherwise, the 72-column title format is used. Figure 2-1 shows examples of the two title formats. The examples are for a DISPLAY_DECK subcommand for base library :$LOCAL.SCD0587.

In general, a listing of names uses the most columns the page width can accommodate.

132-Column Format:

```
DISPLAY_DECK                                NOS/VE SOURCE CODE UTILITY V0.1 82263   1982-09-20    11:27:43   PAGE 2
BASE=:$local.scd0587
```

72-Column Format:

```
DISPLAY_DECK                                                  11:27:43   PAGE 2
1982-09-20          NOS/VE SOURCE CODE UTILITY V0.1 82263
BASE=:$local.scd0587
```

Figure 2-1. Listing Title Formats

Setting Your Working Catalog

You may want to establish a working catalog in your user prolog. By default, NOS/VE establishes $LOCAL as your working catalog when you log in. You can also change your working catalog during interactive sessions with the SET_WORKING_CATALOG command.

Setting your working catalog to $USER eliminates the need to specify the full file path to reference one of your permanent files. To set your working catalog to $USER (the catalog containing your permanent files), enter:

```
/set_working_catalog catalog=$user
```

For example, if you are user PAT in the preceding figure and have set your working catalog to $USER, you can make a reference to file C_PROG as follows:

```
EXAMPLES.C_PROG
```

Another advantage of setting your working catalog to $USER is that any files you created or copied to your $USER catalog are permanent. Suppose you created file MONTHLY_REPORT and your working catalog is set to $LOCAL. Before you log out, you need to copy file MONTHLY_REPORT to your $USER catalog to make it permanent. All files created in $LOCAL are automatically deleted when you log out.

Online Assistance

Any time you are working at your terminal, you can get online assistance from NOS/VE to learn about commands and parameters. One way you can get help is to use the DISPLAY_COMMAND_INFORMATION command.

For example, to find out the parameter requirements for the CREATE_FILE command, enter the following:

```
/display_command_information create_file
```

NOS/VE then displays the names and types for the parameters of the CREATE_FILE command:

```
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 column on the left shows the parameter name and the column on the right shows the data type requirement for each parameter. The column on the right also shows the default value for the parameter (if any) and whether the parameter is required.

For example, the RETENTION parameter is an integer in the range from 1 through 999 and is not required. If you do not specify a value, NOS/VE assigns it the value 999 by default.

The following table summarizes the commands provided by NOS/VE to help you get online information.

Command	Description
DISPLAY_COMMAND_LIST_ ENTRY	Lists all available commands. This comand is especially useful when you are using a utility.
DISPLAY_COMMAND_ INFORMATION	Lists a command's parameters, abbreviations, types, and default values.
DISPLAY_FUNCTION_ INFORMATION	Lists a function's parameters, abbreviations, types, and default values.
HELP	Accesses an online manual containing general information about the subject specified in this command. If no subject is specified and the last NOS/VE command resulted in an error, HELP accesses the online manual that explains the error message. If a utility is initiated, HELP displays a list of the utility's subcommands and functions.
EXPLAIN	Accesses the online manual you specify in this command. If no manual is specified, EXPLAIN accesses the NOS/VE System Information manual.

SCU Commands and Subcommands

This section describes the commands used to manage source libraries.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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)
```

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.

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.

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).

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.

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).

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).

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).

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.

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.

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).

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 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).

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).

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>

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.

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
```

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.

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.

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.

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.

Examples The following subcommand extracts modification MOD1 onto file SAVE_MOD1.

```
sc/extract_modification mod1 save_mod1
```

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.

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.

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
```

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.

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.

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.

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.

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.

Examples The following subcommand writes an intermediate library to the result library file.

```
sc/write_library
```

NOS/VE Commands

The following are NOS/VE commands used with source libraries, listed alphabetically.

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.

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
 $800716132S0209D19880225T220933 Working Library
* SOURCE_LIBRARY                 F$$__00011BD0_E3
```

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.

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
```

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.

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
```

Using the EDIT_FILE Utility 3

Using the EDIT_FILE Utility 3

This chapter explains how to activate the EDIT_FILE utility within an SCU session, and lists the deck selection subcommands used for editing decks. Refer to the NOS/VE File Editor manual for complete details.

Calling the EDIT_FILE Utility

You can use the EDIT_FILE utility to enter and change the contents of a file or deck. To edit a text file, refer to the EDIT_FILE command described in the NOS/VE File Editor manual.

Use the EDIT_DECK subcommand to edit decks in an SCU source library within an SCU session.

After using EDIT_DECK to enter the EDIT_FILE utility, the following subcommands are available to select decks you wish to edit from the working library.

EDIT_DECK
EDIT_FIRST_DECK
EDIT_LAST_DECK
EDIT_NEXT_DECK
END_DECK
RESET_DECK
SELECT_DECK
SELECT_FIRST_DECK
SELECT_LAST_DECK
SELECT_NEXT_DECK

Decks can be edited using all available EDIT_FILE subcommands. Refer to the NOS/VE File Editor manual for complete details.

For further information about the EDIT_DECK subcommand, refer to chapter 2.

You can also use the following SCU subcommands during an editing session.

CREATE_MODIFICATION
DISPLAY_DECK
DISPLAY_DECK_LIST
DISPLAY_DECK_REFERENCES
DISPLAY_FEATURE
DISPLAY_FEATURE_LIST
DISPLAY_GROUP
DISPLAY_GROUP_LIST
DISPLAY_LIBRARY
DISPLAY_MODIFICATION
DISPLAY_MODIFICATION_LIST
EXTRACT_MODIFICATION

Deck Selection Subcommands

The deck selection subcommands offer two ways to get the deck you want, EDIT or SELECT. Subcommands beginning with EDIT open a new deck for editing while maintaining your position in the previous decks (if any). They include:

EDIT_DECK
EDIT_FIRST_DECK
EDIT_LAST_DECK
EDIT_NEXT_DECK

Subcommands beginning with SELECT open a new deck and close the previous decks. They include:

SELECT_DECK
SELECT_FIRST_DECK
SELECT_LAST_DECK
SELECT_NEXT_DECK

The NOS/VE File Editor manual describes the deck selection subcommands.

SCU Text-Embedded Directives 4

SCU Text-Embedded Directives 4

This chapter describes SCU text-embedded directives, presented in alphabetical order.

NOTE

The directive formats in this chapter use the default key character *. The key character for the library is specified when the library is created with the CREATE_LIBRARY command.

Text-embedded directive processing is described in chapter 1.

BLOCK/BLOCKEND

Purpose Marks the beginning and end of a block of text.

Format

***BLOCK**
and
***BLOCKEND**

Remarks

- Blocks determine the range of a conditional copy (refer to the COPYC directive description).
- A BLOCKEND directive must follow each BLOCK directive within the deck. Blocks can be nested up to 32 levels.

Examples The following sequence of lines delimit two blocks, one nested within the other.

```
*block
*copy deck1
*block
*copyc deck2
*blockend
*copy deck3
*blockend
```

When the text is expanded, COPY directives copy DECK1 and DECK3 in the outer block. A COPYC directive copies DECK2 in the inner block.

Another example shows two CYBIL modules expanded from the same deck. In this case, the content of FSP$OPEN_FILE would be copied in both places, since COPYC is in separate blocks.

```
*block
module one;
  :
*copyc fsp$open_file
modend one;
*blockend
*block
module two;
  :
*copyc fsp$open_file
modend two
*blockend
```

COPY

Purpose Copies the text of the specified deck into the expanded text file.

Format ***COPY name**

Parameters **name**

Name of the deck to be copied. This parameter is required.

Examples The COPY directive in the outer block copies DECK1; the COPYC directive does not copy DECK1 because it has already been copied in an enclosing block. Quote marks indicate a comment on the inner deck that is not copied.

```
*block
*copy deck1
*block
*copyc deck1      "Inner DECK1 is not copied.
*blockend
*blockend
```

COPYC

Purpose Conditionally copies the text of the specified deck into the expanded text file. The COPYC directive copies text only if the deck has not already been copied within the current block or within an enclosing block.

Format ***COPYC name**

Parameters

name

Name of the deck to be copied. This parameter is required.

Remarks

- The current block begins with the last BLOCK directive or the beginning of the deck; the current block ends with the next BLOCKEND directive or the end of the deck. A block encloses the current block if its BLOCK directive occurs before the BLOCK directive for the current block and its BLOCKEND directive occurs after the BLOCKEND directive of the current block.

- If SCU expands a deck which tries to COPYC (or COPY) a nonexisting deck, an error message is sent to the listing, an error status is set, and the compile file is generated (without the COPYC or COPY directive).

Examples The following directives delimit three blocks, two inner blocks nested within an outer block.

```
*block
*copyc decka          (Copies the deck)
*block
*copyc decka          (Does not copy the deck)
*copyc deckb          (Copies the deck)
*blockend
*block
*copyc decka          (Does not copy the deck)
*copyc deckb          (Copies the deck)
*blockend
*copyc deckb          (Copies the deck)
*blockend
```

When these directives are processed during text expansion, DECKA is copied only once. The COPYC directive in the outer block copies DECKA, but the COPYC directives in the inner blocks do not copy DECKA because it has already been copied within an enclosing block.

DECKB is copied three times, once in each of the inner blocks and once in the outer block.

DECK

Purpose Specifies the deck name for the following text.

NOTE

The DECK directive is processed when the file containing the directive is specified on the SOURCE parameter of a CREATE_DECK subcommand. The command must also specify that deck directives are to be included. After processing the DECK directive, SCU discards the directive; it does not include the directive in the deck text.

Format

***DECK**
DECK=name
EXPAND=boolean

Parameters

DECK

Deck name. This parameter is required.

EXPAND or *EXP*

Optional expand attribute for the deck.

TRUE

The deck is expanded if specified on an EXPAND_DECK subcommand or a COPY or COPYC directive.

FALSE

The deck is not expanded if specified on an EXPAND_DECK subcommand; it is expanded if specified on a COPY or COPYC directive.

If EXPAND is omitted, TRUE is used.

Examples The following directive names a deck DECK2. Its expand attribute is TRUE.

```
*deck deck2
```

ELSE

Purpose Marks the beginning of a block of text to be expanded if the conditions on the preceding IF directive and any preceding ELSEIF directives are all FALSE.

Format *ELSE

Remarks An IF/IFEND block can contain more than one ELSEIF directive, but only one ELSE directive.

Examples The following sequence of lines contains three blocks of text, only one of which is expanded.

```
*if $deck(DSD)
Text to be expanded if DSD is a deck on the library.
*elseif $deck(DSO)
Text to be expanded if DSO is a deck on the library.
*else
Text to be expanded if neither DSD nor DSO is a deck on the library.
*ifend
```

ELSEIF

Purpose Specifies a condition that is evaluated if the conditions on the preceding IF directive and any preceding ELSEIF directives are all FALSE. The condition determines whether the text between the ELSEIF and the next ELSEIF, ELSE, or IFEND directive is expanded. If the condition on the ELSEIF is TRUE, the text is expanded. If it is FALSE, the text is not expanded.

Format ***ELSEIF boolean expression**

Parameters **boolean expression**

Expression that can be evaluated as either TRUE or FALSE. This parameter is required.

Remarks The boolean expression on the directive is an SCL expression as described in the NOS/VE System Usage manual. It can reference SCL variables. SCL returns an error if the expression is invalid. When SCU detects this error, the entire IF block is suppressed and it does not go to the compile file.

Examples The following text lines comprise an IF/IFEND block subdivided by ELSEIF and ELSE directives.

```
*if $modification(MOD1)
Text expanded if the modification MOD1 has not
been excluded.
*elseif $modification(MOD2)
Text expanded if the modification MOD1 has
been excluded, but the modification
MOD2 has not been excluded.
*else
Text expanded if MOD1 and MOD2 have both
been excluded.
*ifend
```

IF/IFEND

Purpose Delimits a block of text that is conditionally expanded. If SCU evaluates the condition specified on the IF directive as TRUE, it expands the text between the IF directive and the next ELSEIF, ELSE, or IFEND directive. If it evaluates the condition as FALSE, it does not expand the text, and it evaluates the condition on the succeeding ELSEIF directive if one exists within the IF/IFEND block.

Format

***IF boolean expression**
and
***IFEND**

Parameters

boolean expression

Expression that can be evaluated as either TRUE or FALSE. This parameter is required.

Remarks

- The boolean expression on the directive is an SCL expression as described in the NOS/VE System Usage manual. It can reference SCL variables. SCL returns an error if the expression is invalid. When SCU detects this error, the entire IF block is suppressed and it does not go to the compile file.
- IF/IFEND blocks can be nested up to 32 levels.

Examples The following text lines delimit two blocks, one block nested within the other. Text in the outer block is expanded if modification MOD1 is in the working library. Text in the inner block is expanded if modifications MOD1 and MOD2 are in the working library.

```
*if $modification(MOD1)
Text to be expanded if the library contains
modification MOD1.
*if $modification(MOD2)
Text to be expanded if the library contains
modification MOD1 and modification MOD2.
*ifend
*ifend
```

PUT

Purpose Specifies a string expression that is evaluated when the deck is expanded and inserted as a line in the expanded deck text.

Format ***PUT string expression**

Parameters **string expression**

Expression that can be evaluated as a string. This parameter is required.

Remarks The string expression on the directive is an SCL expression as described in the NOS/VE System Usage manual. It can reference SCL string variables and SCL or SCU functions that return string values. The only SCL string operation is concatenation.

Examples The following directive inserts a line in the expanded text consisting of three concatenated strings.

```
*PUT 'The library is '//$library_header(name)//'.'
```

Assuming the name in the library header is MY_LIBRARY, SCU writes the following line when it expands the directive.

```
The library is MY_LIBRARY.
```

TEXT/TEXTEND

Purpose Delimits a block of text in which embedded directives are not processed.

Format ***TEXT**
and
***TEXTEND**

Remarks TEXT/TEXTEND blocks cannot be nested. If a TEXTEND directive does not follow each TEXT directive, SCU sends a warning message when it expands the deck.

Examples The COPY directive in the following text is not processed as a directive, but is copied as a text line because of the preceding TEXT directive.

```
*text
*copy deck1
*textend
```

WEOP

Purpose Writes an end-of-partition delimiter on the expanded text file.

NOTE

Because only the variable (V) record type supports partitioning, the RECORD_TYPE attribute of the file on which SCU writes the text must be V.

Format ***WEOP**

Examples The following example shows a WEOP directive separating two blocks of text.

```
Text for the first partition.
*weop
Text for the second partition.
```

WEOPC

Purpose Conditionally writes an end-of-partition delimiter on the expanded text file. SCU writes the partition delimiter if it has written text since it wrote the previous partition delimiter.

NOTE

Because only the variable (V) record type supports partitioning, the RECORD_TYPE attribute of the file on which SCU writes the text must be V.

Format ***WEOPC**

Examples The following example shows a WEOPC directive that writes an end-of-partition if the previous COPYC directive writes text on the file.

```
Text for the first partition.
*weop
*copyc deck1
*weopc
Text for the second or third partition.
```

Selection Criteria Subcommands 5

Selection Criteria Subcommands 5

Selection criteria processing is useful for expanding or extracting portions of a source library file. Selection criteria subcommands allow you to select decks to be expanded or extracted by including or excluding them based on information contained in the deck headers. For example, the INCLUDE_GROUP subcommand causes all decks in a specific group to be selected.

Selection Criteria Subcommand Processing

To process selection criteria subcommands, specify the SELECTION_CRITERIA parameter on an EXPAND_DECK subcommand, EXTRACT_DECK subcommand, or EXTRACT_SOURCE_LIBRARY command. SCU reads the subcommands to be processed from the selection criteria file specified on the parameter. If the file specified is COMMAND, SCU prompts you for subcommand entry from the command input file with the prompt scc/.

The effect of the selection criteria subcommands entered is cumulative. SCU processes each subcommand as it is entered; however, the effect of subcommands entered earlier is overridden by the most recent subcommand that affects the deck or modification. For example, consider the following subcommand sequence.

```
sc/expand_deck deck1..deck30 selection_criteria=command
scc/exclude_group group1
scc/include_deck deck1
```

The second subcommand includes deck DECK1 even if the deck belongs to group GROUP1. However, if the subcommands are entered in reverse order, DECK1 would not be included if it belongs to group GROUP1.

Due to the cumulative effect in the following subcommands, the second subcommand overrides the first so that all decks that are in GROUP1 will expand, not just the decks in the range of DECK1 to DECK2.

```
sc/expand_deck deck1..deck2 selection_criteria=command
scc/include_group group1
```

You can use SCU functions to loop through the source library from DECK1 to DECK2 to see if a particular deck belongs to GROUP1.

The selection criteria subcommands are valid only during selection criteria processing, which places a command list entry containing the selection criteria subcommands on the NOS/VE command list. The name of the entry is SCU_CRITERIA.

During criteria file processing, NOS/VE searches the command list in restricted mode. Therefore, commands entered in the selection criteria file other than selection criteria subcommands must be prefixed by a /. The prefix notifies NOS/VE that this command is located in a different command list than the one that is currently being used, namely, the selection criteria command list. For more information on restricted command list search mode, refer to the NOS/VE System Usage manual. SCL control statements and assignment statements do not require a prefix.

The following SCU subcommands are allowed during criteria file processing.

DISPLAY_DECK

DISPLAY_DECK_REFERENCES

DISPLAY_FEATURE

DISPLAY_FEATURE_LIST

DISPLAY_LIBRARY

DISPLAY_MODIFICATION

DISPLAY_MODIFICATION_LIST

The following example lists the contents of a selection criteria file.

```
/copy_file input=select_1

  include_modification modification=(mod1,mod3)
  include_group group=section_1
  exclude_deck deck=(deck5..deck7)
  /display_library
  /quit
```

Selection Criteria Subcommands

This section describes selection criteria subcommands, presented in alphabetical order.

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.

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.

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.

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 to be used only for decks copied by COPY and COPYC directives. No other decks on the excluded library are expanded.

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.

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.

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
```

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.

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.

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.

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.

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.

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.

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.

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
```

QUIT
Selection Criteria Subcommand

Purpose Ends SELECTION_CRITERIA_COMMAND command processing.

Format

QUIT or
END or
QUI
STATUS=status variable

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.

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
```

SCU Functions 6

SCU Functions 6

This chapter describes the SCU functions, presented alphabetically.

All function parameters are required. You must supply a value for each parameter shown in the function format.

SCU functions are processed using the SCL interpreter. The functions use SCL condition handling, and follow the SCL command syntax. Refer to the NOS/VE System Usage manual for a complete definition of SCL command syntax.

The $CURRENT_DECK function is an EDIT_FILE function and can only be used within an editing session. Refer to the NOS/VE File Editor manual for more information.

Using SCU Functions

The following example allows you to see if deck MY_DECK is on library SCUPL:

```
/scu
sc/use_library scupl
sc/display_value $deck(my_deck)
TRUE
sc/quit no
```

The following batch example writes all decks from the library on file SCUPL to file COMPILE if the decks are to be processed by CYBIL, were written by John Doe, and have the expand attribute.

```
scu
 use_library scupl
    x=$first_deck
    loop
        if ($deck_header($name(x),processor)='CYBIL') and ..
           ($deck_header($name(x),author)='John Doe') and ..
           ($deck_header($name(x),expand)=true) then
           expand_deck $name(x) compile.$EOI
        ifend
    exit when x=$last_deck
        x=$next_deck($name(x))
    loopend
quit write_library=false
```

The functions $FIRST_DECK, $LAST_DECK, and $NEXT_DECK return strings as their values.

The following example:

- Changes the state of all modifications associated with the feature UPDATE_FEATURE_TO_REVISION_C to 1.
- Produces the detailed display on all modifications associated with feature UPDATE_FEATURE_TO_REVISION_B.
- Raises the state of all modifications currently in state 3 to state 4.
- Writes the result library to file VE1107.

```
scu
 use_library doc180 ve1107
    modnames=$modification_list
    for mods=1 to $variable(modnames,upper_bound) do
        if $modification_header($name(modnames(mods)),feature)= ..
           'UPDATE_FEATURE_TO_REVISION_C' then
           change_modification $name(modnames(mods)) state=1
        ifend
        if $modification_header($name(modnames(mods)),feature)= ..
           'UPDATE_FEATURE_TO_REVISION_B' then
           display_modification $name(modnames(mods)) ..
           display_options=full
        ifend
        if $modification_header($name(modnames(mods)),state)=3 then
           change_modification $name(modnames(mods)) state=4
        ifend
    forend
quit write_library=true
```

The function $MODIFICATION_LIST returns an array of strings as its value.

$BASE
SCU Function

Purpose Returns the base library file.

Format **$BASE**

Parameters None.

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.

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.

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.

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/
```

$ERRORS_FILE
SCU Function

Purpose Returns the file to which intermediate diagnostic messages are written.

Format **$ERRORS_FILE**

Parameters None.

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
```

$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.

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.

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.

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.

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.

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.

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.

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.

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)
```

$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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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.

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)
```

$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.

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
```

Glossary

Glossary A

A

Active Lines

Lines expanded when a deck is expanded. See also Inactive Lines.

Authority

A privilege granted by the application information field of the file permit entry. A digit from 1 through 4 in the field indicates the maximum modification state that the user has authority to change. The letter I in the field grants authority to set an interlock on a deck in the file.

B

Base Library

The library from which the working library is copied. You must have read permission to use this file.

Boolean

A kind of value that is evaluated as TRUE or FALSE.

C

Command Utility

A NOS/VE processor that adds its command table (referred to as its subcommands) to the beginning of the SCL command list. The subcommands are removed from the command list when the processor terminates.

Control Statement

A statement used to structure and control the flow of a job.

Creation Modification

The modification specified on the CREATE_DECK subcommand.

Criteria

See Selection Criteria.

Current Position

The line in the current deck from which the editor determines the location for an operation. The current position line can be referenced with the keyword CURRENT.

D

Deck

A collection of lines with a header describing the collection. For example, a deck can be a compilation unit, a procedure, or a job. You reference a deck by its name.

Deck List

An alphabetical list of the decks on a source library.

Deck Reference

Specification of a deck on a COPY or COPYC text-embedded directive.

Default

The assumed value for a parameter when the parameter is not specified by the user.

Delimiter String

A string that marks the end of text input.

E

Editing Session

The time from when you start the EDIT_FILE utility (the editor) to the time you stop it.

Expand Attribute

A deck header field that determines whether a deck is expanded when specified on an EXPAND_DECK subcommand. For example, if DECKS=DECK1..DECK5 is specified, only the decks in that range that have an expand attribute of TRUE are expanded.

Expansion

Reformatting from the compressed text format on an SCU library to input text format on a text file. Unlike extraction, expansion also processes all text-embedded directives in the deck.

Extraction

Reformatting from the compressed text format on a source library to input text format on a text file. Unlike expansion, extraction does not process text-embedded directives in the deck.

F

Feature

A collection of modifications. A modification can belong to only one feature.

Feature List

A list of all features on a library listed in the order they were added to the library.

File

An SCL element that specifies a temporary or permanent file, including its path and, optionally, a cycle reference (for permanent files). A file is identified by specifying a path and, optionally, a cycle reference (for permanent files) as follows:

path.cycle reference

See also Path and Cycle Reference.

Some files can also be positioned. See also File Position.

File Position

The location in the file at which the next read or write operation will begin. A file that can be positioned is identified by specifying a path, an optional cycle reference (for permanent files), and an optional file position as follows:

path.cycle reference.file position

The file position designators are:

$ASIS	Leave the file in its current position.
$BOI	Position the file at the beginning-of-information.
$EOI	Position the file at the end-of-information.

See also Path and Cycle Reference.

G

Group

A collection of decks in a library. A deck can belong to one or more groups or to no group.

Group List

An alphabetical list of the groups in a library.

I

Inactive Lines

Lines deleted by a modification. Inactive lines are not included when a deck is expanded. Inactive lines become active lines again if the modification that deleted the lines is itself deleted. See also Active Lines.

Integer

A value representing one of the numbers 0, +1, -1, +2, -2, and so forth.

Integer Constant

One or more digits and, for hexadecimal integer constants, the following characters:

A B C D E F a b c d e f

A hexadecimal integer constant must begin with a digit. A preceding sign and subsequent radix are optional.

Interlock

A field set in the deck header indicating that the deck has been extracted. The original interlock field is set in the deck header on the extracted library; the subinterlock field is set in the deck header on the base library. You must have the correct authority to set this field. See also Authority.

Intermediate Diagnostic Message

Diagnostic message issued by a command. The command continues processing after the message. When command processing completes, the command returns another diagnostic message.

K

Key Character

A character that prefixes each text-embedded directive in a source library.

Keyword

A parameter value that has special meaning in the context of a particular parameter. For example, a parameter called COUNT might normally expect an integer but could be given the keyword ALL.

L

Last Used Deck

The last deck name explicity entered for a DECK parameter. The last used deck is also the default for most subcommands with a DECK parameter.

Last Used Modification

The last modification name explicity entered for a MODIFICATION parameter. The last used modification is also the default for most subcommands with a MODIFICATION parameter.

Line

SCU recognizes a line as a record of text.

A sequence of characters. SCU assigns a unique identifier to each line so you can reference individual lines.

Line Identifier

The unique identifier of a line in a deck. The line identifier consists of a modification name followed by a sequence number. The modification name identifies the modification to which the line belongs.

List

A command format notation specifying that a parameter can be given several values.

M

Modification

A collection of line changes with a header describing the collection.

Modification List

A list of the modifications on a library. It is ordered alphabetically.

N

Name, SCL

Combination of from 1 through 31 characters chosen from the following set:

- Alphabetic characters (A through Z and a through z).
- Digits (0 through 9).
- Special characters: #, @, $, _, [,], \, ^, `, {, }, |, ~.

The first character of a name cannot be numeric.

P

Partition

A unit of data in a sequential or byte-addressable file delimited by end-of-partition separators or the beginning- or end-of-information.

R

Range

Value represented as two values separated by an ellipsis. The element is associated with the values from the first value through the second value. The first value must be less than or equal to the second value. For example:

```
value..value
```

Result Library

The file to which the working library can be copied. You must have read and write permission to use this file. An SCU session always writes the working library to the result library file unless the WRITE_LIBRARY parameter on the QUIT command directs it not to do so.

S

SCL Variable

The means of storing a value to be tested or displayed by an SCL statement.

SCU Session

The period of time from when you start the Source Code Utility to when you stop it.

Selection Criteria

Qualifications specified by selection criteria subcommands for the decks and modifications expanded or extracted.

Sequencing

The process that assigns new line identifiers to the lines in a deck or modification.

Source Library

A collection of decks on a file, with a header describing the collection, generated and manipulated by the Source Code Utility (SCU).

You reference a source library by the file on which it resides.

State

A field in a modification header indicating the current development level of the modification.

Status Variable

A variable record of kind status that holds the completion status of a command.

String

A value that represents a sequence of characters.

String Constant

A sequence of characters delimited by apostrophes ('). An apostrophe can be included in the string by specifying two consecutive apostrophes.

T

Text-Embedded Directive

A text line that SCU processes as a directive when expanding a deck or a file.

V

Value List

A series of value sets separated by spaces or commas and enclosed in parentheses. If only one value set is given in the list, the parentheses can be omitted. For example:

(value set,value set,value set)

or

value set

See also Value, Value Element, and Value Set.

Value Set

A series of value elements separated by spaces or commas and enclosed in parentheses. If only one value element is given in the set, the parentheses can be omitted. For example:

(value element,value element,value element)

or

value element

See also Value, Value Element, and Value List.

W

Working Library

The temporary library on which an SCU session performs its operations. It is copied from the base library and can be copied to the result library.

Related Manuals B

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 B

All NOS/VE manuals and related hardware manuals are listed in table B-1. If your site has installed the online manuals, you can find an abstract for each NOS/VE manual in the online System Information manual. To access this manual, enter:

```
/explain
```

Ordering Printed Manuals

To order a printed Control Data manual, send an order form to:

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

To obtain an order form or to get more information about ordering Control Data manuals, write to the above address or call (612) 292-2101. If you are a Control Data employee, call (612) 292-2100.

Accessing Online Manuals

To access the online version of a printed manual, log in to NOS/VE and enter the online title on the EXPLAIN command (table B-1 supplies the online titles). For example, to see the NOS/VE Commands and Functions manual, enter:

```
/help manual=scl
```

The examples in some printed manuals exist also in the online Examples manual. To access this manual, enter:

```
/help manual=examples
```

When EXAMPLES is listed in the Online Manuals column in table B-1, that manual is represented in the online Examples manual.

Table B-1. Related Manuals

Manual Title	Publication Number	Online 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 B-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 B-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 B-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 B-1. Related Manuals *(Continued)*

Manual Title	Publication Number	Online Manuals[1]
FORTRAN Manuals:		
FORTRAN Version 1 for NOS/VE Language Definition Usage	60485913	EXAMPLES
FORTRAN Version 1 for NOS/VE Quick Reference		FORTRAN
FORTRAN Version 2 for NOS/VE Language Definition Usage	60487113	EXAMPLES
FORTRAN Version 2 for NOS/VE Quick Reference		VFORTRAN
FORTRAN for NOS/VE Tutorial	60485912	FORTRAN_T
FORTRAN for NOS/VE Topics for FORTRAN Programmers Usage	60485916	
FORTRAN for NOS/VE Summary	60485919	
COBOL Manuals:		
COBOL for NOS/VE Summary	60486019	

1. This column lists the title of the online version of the manual and indicates whether the examples in the printed manual are in the online Examples manual.

(Continued)

Table B-1. Related Manuals *(Continued)*

Manual Title	Publication Number	Online 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 B-1. Related Manuals *(Continued)*

Manual Title	Publication Number	Online Manuals[1]
Other Compiler Manuals (Continued):		
Pascal for NOS/VE Usage	60485613	PASCAL and EXAMPLES
Prolog for NOS/VE Quick Reference	60486718	PROLOG
Prolog for NOS/VE Usage	60486713	
VX/VE Manuals:		
C/VE for NOS/VE Quick Reference		C
C/VE for NOS/VE Usage	60469830	
DWB/VX Introduction and User Reference Tutorial/Usage	60469890	
DWB/VX Macro Packages Guide Usage	60469910	
DWB/VX Preprocessors Guide Usage	60469920	
DWB/VX Text Formatters Guide Usage	60469900	

1. This column lists the title of the online version of the manual and indicates whether the examples in the printed manual are in the online Examples manual.

(Continued)

Table B-1. Related Manuals *(Continued)*

Manual Title	Publication Number	Online 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 B-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 B-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 B-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 B-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 B-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 B-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 B-1. Related Manuals *(Continued)*

Manual Title	Publication Number	Online Manuals[1]
Hardware Manuals (Continued):		
CYBER 170 Computer Systems, Models 815, 825, 835, 845, and 855 CYBER 180 Models 810, 830, 835, 840, 845, 850, 855, and 860 Maintenance Register Codes Booklet	60458110	
HPA/VE Reference	60461930	
Virtual State Volume II Hardware Reference	60458890	
7021-31/32 Advanced Tape Subsystem Reference	60449600	
7221-1 Intelligent Small Magnetic Tape Subsystem Reference	60461090	

1. This column lists the title of the online version of the manual and indicates whether the examples in the printed manual are in the online Examples manual.

ASCII Character Set

This appendix lists the ASCII character set (refer to table C-1).

NOS/VE supports the American National Standards Institute (ANSI) standard ASCII character set (ANSI X3.4-1977). NOS/VE represents each 7-bit ASCII code in an 8-bit byte. These 7 bits are right justified in each byte. For ASCII characters, the eighth or leftmost bit is always zero. However, in NOS/VE the leftmost bit can also be used to define an additional 128 characters.

If you want to define additional non-ASCII characters, be certain that the leftmost bit is available in your current working environment. The full screen applications (such as the EDIT_FILE utility, the EDIT_CATALOG utility, and the programming language environments) already use this bit for special purposes. Therefore, these applications accept only the standard ASCII characters. In applications in which the leftmost bit is not used, however, you are free to use it to define the interpretation of each character as you wish.

Table C-1. ASCII Character Set

Decimal Code	Hexa-decimal Code	Octal Code	Graphic or Mnemonic	Name or Meaning
000	00	000	NUL	Null
001	01	001	SOH	Start of heading
002	02	002	STX	Start of text
003	03	003	ETX	End of text
004	04	004	EOT	End of transmission
005	05	005	ENQ	Enquiry
006	06	006	ACK	Acknowledge
007	07	007	BEL	Bell
008	08	010	BS	Backspace
009	09	011	HT	Horizontal tabulation
010	0A	012	LF	Line feed
011	0B	013	VT	Vertical tabulation
012	0C	014	FF	Form feed
013	0D	015	CR	Carriage return
014	0E	016	SO	Shift out
015	0F	017	SI	Shift in
016	10	020	DLE	Data link escape
017	11	021	DC1	Device control 1
018	12	022	DC2	Device control 2
019	13	023	DC3	Device control 3
020	14	024	DC4	Device control 4
021	15	025	NAK	Negative acknowledge
022	16	026	SYN	Synchronous idle
023	17	027	ETB	End of transmission block
024	18	030	CAN	Cancel
025	19	031	EM	End of medium
026	1A	032	SUB	Substitute
027	1B	033	ESC	Escape
028	1C	034	FS	File separator
029	1D	035	GS	Group separator
030	1E	036	RS	Record separator
031	1F	037	US	Unit separator
032	20	040	SP	Space
033	21	041	!	Exclamation point
034	22	042	"	Quotation marks
035	23	043	#	Number sign
036	24	044	$	Dollar sign
037	25	045	%	Percent sign
038	26	046	&	Ampersand
039	27	047	'	Apostrophe
040	28	050	(	Opening parenthesis
041	29	051	)	Closing parenthesis
042	2A	052	*	Asterisk
043	2B	053	+	Plus

(Continued)

Table C-1. ASCII Character Set (Continued)

Decimal Code	Hexa-decimal Code	Octal Code	Graphic or Mnemonic	Name or Meaning
044	2C	054	,	Comma
045	2D	055	-	Hyphen
046	2E	056	.	Period
047	2F	057	/	Slant
048	30	060	0	Zero
049	31	061	1	One
050	32	062	2	Two
051	33	063	3	Three
052	34	064	4	Four
053	35	065	5	Five
054	36	066	6	Six
055	37	067	7	Seven
056	38	070	8	Eight
057	39	071	9	Nine
058	3A	072	:	Colon
059	3B	073	;	Semicolon
060	3C	074	<	Less than
061	3D	075	=	Equals
062	3E	076	>	Greater than
063	3F	077	?	Question mark
064	40	100	@	Commercial at
065	41	101	A	Uppercase A
066	42	102	B	Uppercase B
067	43	103	C	Uppercase C
068	44	104	D	Uppercase D
069	45	105	E	Uppercase E
070	46	106	F	Uppercase F
071	47	107	G	Uppercase G
072	48	110	H	Uppercase H
073	49	111	I	Uppercase I
074	4A	112	J	Uppercase J
075	4B	113	K	Uppercase K
076	4C	114	L	Uppercase L
077	4D	115	M	Uppercase M
078	4E	116	N	Uppercase N
079	4F	117	O	Uppercase O
080	50	120	P	Uppercase P
081	51	121	Q	Uppercase Q
082	52	122	R	Uppercase R
083	53	123	S	Uppercase S
084	54	124	T	Uppercase T
085	55	125	U	Uppercase U
086	56	126	V	Uppercase V
087	57	127	W	Uppercase W

(Continued)

Table C-1. ASCII Character Set (Continued)

Decimal Code	Hexa-decimal Code	Octal Code	Graphic or Mnemonic	Name or Meaning
088	58	130	X	Uppercase X
089	59	131	Y	Uppercase Y
090	5A	132	Z	Uppercase Z
091	5B	133	[	Opening bracket
092	5C	134	\	Reverse slant
093	5D	135	]	Closing bracket
094	5E	136	^	Circumflex
095	5F	137	_	Underline
096	60	140	`	Grave accent
097	61	141	a	Lowercase a
098	62	142	b	Lowercase b
099	63	143	c	Lowercase c
100	64	144	d	Lowercase d
101	65	145	e	Lowercase e
102	66	146	f	Lowercase f
103	67	147	g	Lowercase g
104	68	150	h	Lowercase h
105	69	151	i	Lowercase i
106	6A	152	j	Lowercase j
107	6B	153	k	Lowercase k
108	6C	154	l	Lowercase l
109	6D	155	m	Lowercase m
110	6E	156	n	Lowercase n
111	6F	157	o	Lowercase o
112	70	160	p	Lowercase p
113	71	161	q	Lowercase q
114	72	162	r	Lowercase r
115	73	163	s	Lowercase s
116	74	164	t	Lowercase t
117	75	165	u	Lowercase u
118	76	166	v	Lowercase v
119	77	167	w	Lowercase w
120	78	170	x	Lowercase x
121	79	171	y	Lowercase y
122	7A	172	z	Lowercase z
123	7B	173	{	Opening brace
124	7C	174	|	Vertical line
125	7D	175	}	Closing brace
126	7E	176	~	Tilde
127	7F	177	DEL	Delete

Conversion Aids D

Conversion Aids D

This appendix describes the SCL commands that convert a source code library in another format to SCU format. Commands exist to convert source libraries in Update format, Modify format, and SCU version 1.0 format.

NOTE

Each conversion command assumes that the file to be converted has been transferred from a NOS or NOS/BE system that uses a 60-bit word and that the transfer has right-justified each 60-bit word in the 64-bit NOS/VE word. To do so, the GET_FILE command to transfer the file from the NOS or NOS/BE system must specify DATA_CONVERSION=B60.

Besides converting the file format, a conversion also changes the file contents as follows:

- The state of each modification is changed to 0. This allows you to change information in the modification header.
- Each WEOR (write end of record) and WEOF (write end of file) text-embedded directive is changed to a WEOP directive. For example, *WEOR,15 becomes *WEOP "15. (NOS/VE has only one level of file separator.)
- The conversion adds a modification named SCU$ALTER to the library if the conversion requires changes to text lines to preserve the logic of text-embedded directives. A full listing for the conversion lists the changed text lines.

Identifier Conversion

The conversion process preserves identifiers used in the Update or Modify file if the names are valid SCL names. A NOS/VE name is valid if it contains only the following:

A through Z
a through z
0 through 9
#,@,$,_,[,],\,^,{,},|,~

The first character of a name cannot be a digit.

If you attempt to convert a file without specifying substitutions for all invalid identifiers, the conversion utility returns a list of invalid identifiers for which substitutions should be specified in the next conversion attempt. The identifiers could name decks, correction sets, and variables. The utility does not attempt to convert the library format until it has a complete list of valid identifiers for the library.

To specify substitute names, you must specify a file of substitutions on the NAME_LIST parameter on the conversion command.

Each line of the file specifies the name to be replaced and one or two replacement names. The line can specify a short name and/or a long name. The short replacement name is used if the name to be replaced occurs as a correction set name; the long replacement name is used if the name to be replaced occurs as a deck name.

The file contains an SCL parameter list for each substitution. The parameters can be specified with or without keywords. Each parameter list must include the OLD_NAME parameter and one or both of the substitution name parameters.

OLD_NAME=name
NEW_NAME=name
MODIFICATION_NAME=name

OLD_NAME or **ON**

Name to be replaced. If the name contains invalid characters, it must be specified as a string (within quotes or apostrophes). This parameter is required.

NEW_NAME or *NN*

New deck name (from 1 through 31 characters). If NEW_NAME is omitted, the name specified by MODIFICATION_NAME is used for deck names. Either NEW_NAME or MODIFICATION_NAME must be specified.

MODIFICATION_NAME

New modification name (from 1 through 9 characters). If MODIFICATION_NAME is omitted, the correction set name specified by OLD_NAME remains unchanged. (Correction set names are not converted to the NEW_NAME name.)

For example, the following file line replaces the Update correction set name MOD**1 with the SCU modification name MOD_1.

```
'mod**1',,mod_1
```

You may want to convert deck names or SCL variable names to longer, more descriptive names. This is possible because names can be up to 31 characters long under NOS/VE. (Modification names must remain a maximum of 9 characters long.)

For example, the following file line replaces the deck name GETREC with the SCU deck name GET_NEXT_RECORD.

```
getrec,get_next_record
```

Update Format Conversion

The CONVERT_UPDATE_TO_SCU command converts a source library from Update format to SCU format.

The Update file can be in sequential format; it cannot be in random format. It must use either 64-character, 6-bit display code or 128-character, 8/12 ASCII code.

NOTE

The conversion utility only converts files that use the 64-character set; it does not convert files that use the 63-character set. Before attempting to convert a file using the 63-character set, convert it to the 64-character set using the NOS command FCOPY.

All inactive lines (lines deleted by YANK, SELYANK, or YANKDECK directives) are discarded during the conversion. In other words, the YANK, SELYANK, and YANKDECK directives are processed as PURGE, SELPURGE, and PURDECK directives, respectively.

Because SCU has no directives corresponding to the following Update directives, the conversion sends a warning message for each and includes the directive in the library as a text line.

Directive	Function
*DEFINE	Defines a condition to be tested by an IF statement (refer to Selection Criteria File).
*DO or *DONT	Overrides deck and correction set yanks during a deck expansion. SCU does not have this feature.
*WIDTH	Overrides the control statement line width specification. SCU allows you to specify a line width for the entire expansion or for each deck expanded, but not within the expanded deck.

Selection Criteria File

If you specify a file on the SELECTION_CRITERIA parameter, the command converts each active DEFINE directive in the YANK$$$ deck to a CREATE_VARIABLE command on the selection criteria file. Because the IF directive in the source text that tested the DEFINE condition is converted to an IF SCU directive, the content of the created SCL variable can be tested by the converted IF directive.

NOTE

If the IF condition is not directly convertible because of overlapping nested IF conditions, the conversion sends a warning message and includes the IF directive as a text line.

The conversion ignores all inactive lines in the YANK$$$ deck. All YANK and SELYANK directives in the YANK$$$ deck are processed as PURGE and SELPURGE directives and are not converted.

NOS/VE Conversion Commands

The following are NOS/VE commands that convert a NOS source code library to a NOS/VE source code library.

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.

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
```

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.

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.

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
```

Maximum Limits for a Source Library E

Maximum Limits for a Source Library E

This appendix lists the maximum limits that apply to an SCU source library.

Description	Limit
Maximum number of decks on a library	262,143
Maximum description size (in characters) for a deck, modification, or library	65,535
Maximum number of modifications on a library	262,143
Maximum number of modifications per deck	16,383
Maximum number of groups	32,767
Maximum number of groups per deck	255
Maximum number of features	65,535
Maximum sequence number	16,777,214
Maximum number of lines per deck	16,777,214

Accessing Online Examples F

Accessing Online Examples F

An online manual named Examples contains examples which show you how to use various NOS/VE concepts, SCL commands, and CYBIL procedures. You can use the online Examples manual to perform the following operations.

- Access examples by name, manual, command name, or procedure name.
- View the example.
- Print the example.
- Copy the example into your $USER catalog for subsequent execution.

To access the online manual, enter:

```
/help manual=examples
```

In response, the system displays a menu of the topics for which examples are provided. This menu includes topics from the following manuals:

COBOL for NOS/VE
CYBIL File Management
CYBIL Keyed-File and Sort/Merge Interfaces
CYBIL Language Definition
CYBIL Sequential and Byte-Addressable Files
CYBIL System Interface
FORTRAN for NOS/VE
Introduction to NOS/VE
NOS/VE File Editor
NOS/VE Screen Formatting
NOS/VE System Usage
NOS/VE Object Code Management
NOS/VE Source Code Management

Accessing Examples by Name or by Manual

In each of the printed manuals containing examples, the example's name is supplied in the introduction to the example. Because the online Examples manual is indexed by example name, you can access the example directly by specifying its name.

For example, suppose you are reading the CREATE_PERMIT_PF_1 example in the CYBIL File Management manual and you want to have a copy of the example in one of your catalogs. You can quickly access the example by using either of the following methods.

- Specify the name of the example on the SUBJECT parameter of the HELP command when you access the manual. For example:

```
help subject=create_permit_pf_1 manual=examples
```

- If you have already accessed the Examples manual, enter the example's name followed by a question mark:

```
create_permit_pf_1?
```

You are then positioned to the introductory screen of the CREATE_PERMIT_PF_1 example. This screen prompts you to view, copy, or print the example.

To access examples associated with a specific manual, select an option from the main menu. The system displays a list of example names associated with that manual. You can then choose a specific example from the list.

Searching for Examples by Command or Procedure Name

The online Examples manual also enables you to search for examples by SCL command or CYBIL procedure names. You can either view the list of index topics by pressing the key associated with the Index operation, or you can access a topic directly by entering the command or procedure name itself.

For example, if you want to look at one or more ways in which the CREATE_FILE command is used, enter the following request on the home line:

```
create_file?
```

If you want to see one or more ways that the FSP$OPEN_FILE procedure call is used in examples, enter:

```
fsp$open_file?
```

In response, the system displays an example that illustrates the use of the procedure or command you specified.

You can also specify the command or procedure name on the SUBJECT parameter of the HELP command when you access the manual. For example:

```
help subject=fsp$open_file manual=examples
```

To view a further example that illustrates the use of the command or procedure you specified, enter another question mark (?). You can enter as many question marks as there are examples indexed for that command or procedure.

When the number of examples for that command or procedure is exhausted, an informative message is displayed.

Viewing, Copying, and Printing an Example

After you access a particular example, the following menu of options appears:

```
Enter your menu choice:
 a. view the example
 b. copy the example
 c. print the example
```

USe the menu of options as follows:

- To view the example, choose menu selection A, followed by a return. The example is displayed at your terminal. Since the example appears in full-screen mode, you can easily move from screen to screen by following the function key prompts.
- To copy the example to a file, choose menu selection B, followed by a return. You are then prompted for the name of the file to which you want the example copied. Once you enter a file name, NOS/VE displays a message verifying the name of the file to which the example was copied.
- To print the example, choose menu selection C. A message soon appears which indicates that the file has been sent to the printer.

Executing an Example

After copying an example to a file, you can easily execute the example by completing the following steps:

1. Exit the online Examples manual by entering a QUIT directive on the home line.
2. Enter the full path name of the file to which the example was copied.

For example, to execute the example contained in file DUP_FILE_EXAMPLE in your $USER catalog, exit the online Examples manual and enter:

```
/$user.dup_file_example
```

Using Function Keys and Directives

Once you access the online Examples manual, you can read it by pressing function keys or by entering directives on the home line.

Function key prompts for using this manual are displayed at the bottom of your screen, provided you are in full-screen mode. These function keys vary according to the type of terminal you are using.

If you need assistance on what a particular function key does, press the help key for your terminal, and then press the function key in question. Pressing the help key again displays a menu of online help options (such as how to use the menus, or how to page forward and backward).

The following function key prompts help you search for examples:

Function Key Prompt	Description
Find	Enables you to locate screens where an example, command, or procedure you specify appears.
Index	Enables you to access the manual's index. After pressing the key associated with this operation, you can do one of the following: • Specify the topic where you want to begin reading the index. • Press RETURN to display the beginning of the index.

Many terminals have function keys or dedicated keys that return you to the main menu (the first screen in the manual). On a VT220 terminal, hold down the shift key and press the **F17** key. Alternatively, you can enter the FIRST or TOP directive on the home line of any terminal at which you can read online manuals.

The Quit function key prompt is associated with the key(s) you press to leave the Examples manual. On a VT220 terminal, press the **F11** key. Alternatively, you can enter the QUIT directive on the home line of any terminal at which you can read online manuals.

Index

Index

D

E

F

G

I

S

T

U

W

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

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.