VAX Language-Sensitive Editor and VAX Source Code Analyzer Reference Manual

Order Number: AA–PAJKA–TK

December 1989

This manual describes the commands of both the VAX Language-Sensitive Editor and the VAX Source Code Analyzer and provides other reference information.

Revision/Update Information:	This document is a new manual.
Operating System and Version:	VMS Version 5.1 or higher for LSE VMS Version 5.2 or higher for SCA
Software Version:	VAX Language-Sensitive Editor Version 3.0 VAX Source Code Analyzer Version 2.0

digital equipment corporation
maynard, massachusetts

First Printing, December 1989

ZK5340

Contents

Command Descriptions

Figures

Tables

Preface

This manual contains reference material on the VAX Language-Sensitive Editor and the VAX Source Code Analyzer.

Intended Audience

This manual is for experienced programmers, technical writers, and technical managers.

Document Structure

The *VAX Language-Sensitive Editor and VAX Source Code Analyzer Reference Manual* has a command dictionary and appendixes that contain reference information.

- The Command Dictionary contains an alphabetical list of all the LSE and SCA commands that are available from command line mode.
- Appendix A describes how to access LSE from other VMS applications.
- Appendix B describes how to access SCA from other VMS applications.
- Appendix C provides information on writing your own VAXTPU procedures.
- Appendix D contains information of interest to VAX FORTRAN and VAX COBOL programmers.
- Appendix E describes how to write your own VAXTPU routines for use with the package facility.
- Appendix F contains a list of the EVE commands with the corresponding LSE commands.

- Appendix G contains information about using the VMS Terminal Fallback Facility to translate double-angle brackets to single-angle brackets on 7-bit terminals.

Associated Documents

- The *VAX Language-Sensitive Editor and VAX Source Code Analyzer User Manual* contains tutorial information on using the VAX Language-Sensitive Editor and the VAX Source Code Analyzer.
- The *VAX Language-Sensitive Editor Installation Guide* contains instructions for installing LSE on VMS operating systems.
- The *VAX Source Code Analyzer Installation Guide* contains instructions for installing SCA on VMS operating systems.
- The *VAX Text Processing Utility Reference Manual* describes the VAX Text Processing Utility features, including the high-level procedural language available for use with LSE.
- *Using VAXset* describes how to use the VAXset products with other VMS software development facilities to create an effective development environment.

Conventions

Convention	Description
$\boxed{\text{RETURN}}$	In interactive examples, a label enclosed in a box indicates that you press a key on the terminal, for example, $\boxed{\text{RETURN}}$.
CTRL/*x*	The phrase CTRL/*x* indicates that you must press the ke labeled CTRL while you simultaneously press another key, for example, CTRL/Y, CTRL/Z, CTRL/G.
KP*n*	The phrase KP*n* indicates that you must press the key labeled with the number or character *n* on the numeric keypad, for example, KP6, KP3.
$ LSEDIT	Interactive examples show all output lines or prompting characters that the system prints or displays in black letters. All user-entered commands are shown in red letters.

Convention	Description
file-spec, . . .	A horizontal ellipsis following a parameter, option, or value in syntax descriptions indicates the additional parameters, options, or values you can enter.
. . .	A horizontal ellipsis in a figure or example indicates that not all of the statements are shown.
. . .	A vertical ellipsis indicates the omission of items from a code example or command format; the items are omitted because they are not important to the topic being discussed.
( )	In format descriptions, parentheses indicate that, if you choose more than one option, you must enclose the choices in parentheses.
[ ]	In format descriptions, brackets indicate that whatever is enclosed is optional; you can select none, one, or all of the choices.
{}	In format descriptions, braces surround a required choice of options; you must choose one of the options listed.
boldface text	Boldface text represents the introduction of a new term.
italic text	Italic text represents parameters, arguments, and information that can vary in system messages (for example, Internal error *number*), as well as book titles.
user input	The hardcopy version of this manual has interactive examples that show user input in red letters and system responses or prompts in black letters. The online version differentiates user input from system responses or prompts by using a different font.
UPPERCASE TEXT	Uppercase letters indicate the name of a command, a routine, the name of a file, the name of a file protection code, or the abbreviation for a system privilege.
mouse	The term *mouse* refers to any pointing device, such as a mouse, a puck, or a stylus.
MB1,MB2,MB3	MB1 indicates the left mouse button, MB2 indicates the middle mouse button, and MB3 indicates the right mouse button. (You can redefine the buttons.)

Command Dictionary

This dictionary describes all of the Language-Sensitive Editor (LSE) and Source Code Analyzer (SCA) commands. Section 1 describes how to enter command mode from your editing session. Section 2 discusses how to cancel commands and return to your editing session. Section 3 discusses how to execute commands typed into a buffer. Section 4 groups LSE and SCA commands by function. The remainder of the dictionary contains the individual command descriptions.

Note that if a section, such as Qualifiers or Parameters, is not applicable to a specific command, the section does not appear under that command.

1 Executing Commands

To execute only one command, enter command mode by pressing the Do key, or use the PF1 and COMMAND (KP7) sequence. Type the command at the LSE Command> prompt and press the Return key. The command executes and LSE returns to keypad mode. In DECwindows only, you can also enter command mode by clicking MB1 below the status line.

LSE provides multiple command recall; by using the up and down arrow keys at the LSE Command> prompt, you can recall any of the commands you issued during your current editing session.

To execute several commands, enter command mode by pressing CTRL/Z. Type the first command at the LSE> prompt and terminate the command string by pressing the Return key. You will still be at the LSE> prompt after the command executes. Press CTRL/Z or enter the CONTINUE command to return to keypad mode.

2 Canceling Commands

To cancel a command, press CTRL/Z in response to a prompt. For example, pressing CTRL/Z in response to the Search for: prompt cancels the SEARCH command. Pressing CTRL/Z in response to the LSE> prompt returns you to keypad editing.

Pressing CTRL/C while the REPEAT or DO/BUFFER command is executing terminates that command.

3 Commands in Buffers

You can execute commands that have been typed into a buffer. At the LSE> prompt, issue the DO command with the /BUFFER qualifier and supply the name of the buffer containing the commands you want (see the individual command descriptions for more information).

4 Command Categories

The following lists identify related LSE and SCA commands and tasks. If you need more information on a command, see its individual description.

4.1 Editing Session Control Commands

ATTACH	RECALL
CONTINUE	REPEAT
DCL	SET DEFAULT_DIRECTORY
DO	SET FONT
EXIT	SET JOURNALING
GOTO COMMAND	SET NOJOURNALING
QUIT	SPAWN

4.2 Text Manipulation Commands

CANCEL SELECT_MARK	PASTE
CAPITALIZE WORD	QUOTE
CENTER LINE	SELECT ALL
CHANGE CASE	SET AUTO_ERASE
CHANGE DIRECTION	SET FORWARD
CHANGE INDENTATION	SET INSERT
CHANGE TEXT_ENTRY_MODE	SET NOAUTO_ERASE
CUT	SET NOWRAP
ENTER LINE	SET OVERSTRIKE
ENTER SPACE	SET REVERSE
ENTER SPECIAL	SET SELECT_MARK
ENTER TAB	SET WRAP
ENTER TEXT	SPELL
ERASE CHARACTER	SUBSTITUTE
ERASE LINE	TAB
ERASE PLACEHOLDER	TOGGLE SELECT_MARK
ERASE SELECTION	UNERASE
ERASE WORD	UNEXPAND
EXPAND	UNTAB
FILL	UPPERCASE WORD
LOWERCASE WORD	

4.3 Entering Source Code Commands

ENTER COMMENT
ENTER PSEUDOCODE
ERASE PLACEHOLDER
EXPAND
GOTO PLACEHOLDER
SET AUTO_ERASE
SET LANGUAGE
SET NOAUTO_ERASE
SET NOLANGUAGE
UNDO ENTER COMMENT
UNERASE
UNEXPAND

4.4 SCA Navigation Commands

EXPAND
GOTO SOURCE
NEXT OCCURRENCE
NEXT STEP
NEXT SYMBOL
PREVIOUS OCCURRENCE
PREVIOUS STEP
PREVIOUS SYMBOL
UNEXPAND

4.5 SCA Library Commands

ANALYZE
CONVERT LIBRARY
CREATE LIBRARY
DELETE LIBRARY
DELETE MODULE
EXTRACT MODULE
LOAD
REORGANIZE
SET LIBRARY
SET NOLIBRARY
SHOW LIBRARY
SHOW MODULE
VERIFY

4.6 SCA Query Commands

FIND
GOTO DECLARATION
INSPECT

4.7 Query Session Manipulation Commands

DELETE QUERY	PREVIOUS QUERY
GOTO QUERY	SHOW QUERY
NEXT QUERY	

4.8 Commands for Compiling Source Code and Reviewing Errors

COMPILE	NEXT STEP
END REVIEW	PREVIOUS ERROR
GOTO REVIEW	PREVIOUS STEP
GOTO SOURCE	REVIEW
NEXT ERROR	

4.9 Indenting Source Code Commands

ALIGN	SET TAB_INCREMENT
CHANGE INDENTATION	SET WRAP
ENTER TAB	TAB
FILL	UNTAB
SET INDENTATION	

4.10 Cursor Movement Commands

CANCEL MARK	GOTO TOP
CHANGE DIRECTION	GOTO WORD
GOTO BOTTOM	LINE
GOTO BUFFER	NEXT ERROR
GOTO CHARACTER	PREVIOUS ERROR
GOTO FILE	SEARCH
GOTO LINE	SET CURSOR
GOTO MARK	SET FORWARD
GOTO PAGE	SET MARK
GOTO PLACEHOLDER	SET REVERSE
GOTO SCREEN	SET SEARCH
GOTO SOURCE	

4.11 Screen Manipulation Commands

CHANGE WINDOW_MODE	OTHER WINDOW
DELETE WINDOW	PREVIOUS WINDOW
ENLARGE WINDOW	REFRESH
GOTO BUFFER	SET SCREEN
GOTO FILE	SET SCROLL_MARGINS
GOTO SCREEN	SHIFT
GOTO SOURCE	SHRINK WINDOW
NEXT WINDOW	SPLIT WINDOW
ONE WINDOW	TWO WINDOWS

4.12 File and Buffer Manipulation Commands

CHANGE DIRECTION
CHANGE TEXT_ENTRY_MODE
CLOSE BUFFER
CUT
DELETE BUFFER
GOTO BUFFER
GOTO FILE
GOTO SOURCE
INCLUDE
NEXT BUFFER
PASTE
PREVIOUS BUFFER
READ
RECOVER BUFFER
SET DEFAULT_DIRECTORY
SET DIRECTORY
SET FORWARD
SET INDENTATION
SET INSERT
SET LEFT_MARGIN
SET MODIFY
SET NOMODIFY
SET NOSOURCE_DIRECTORY
SET OUTPUT_FILE
SET OVERSTRIKE
SET READ_ONLY
SET REVERSE
SET RIGHT_MARGIN
SET SOURCE_DIRECTORY
SET TAB_INCREMENT
SET WRAP
SHOW BUFFER
WRITE

4.13 Program Design Commands

COLLAPSE
DEFINE ADJUSTMENT
DEFINE KEYWORDS
DEFINE TAG
DELETE ADJUSTMENT
DELETE KEYWORDS
DELETE TAG
ENTER COMMENT
ENTER PSEUDOCODE
EXPAND
EXTRACT ADJUSTMENT
EXTRACT KEYWORDS
EXTRACT TAG
FOCUS
REPORT
SET NOOVERVIEW
SET OVERVIEW
SHOW ADJUSTMENT
SHOW KEYWORDS
SHOW TAG
UNDO ENTER COMMENT
VIEW SOURCE

4.14 Commands for Tailoring the Environment

- CALL
- CHECK LANGUAGE
- DEFINE ADJUSTMENT
- DEFINE ALIAS
- DEFINE COMMAND
- DEFINE KEY
- DEFINE KEYWORDS
- DEFINE LANGUAGE
- DEFINE PACKAGE
- DEFINE PARAMETER
- DEFINE PLACEHOLDER
- DEFINE ROUTINE
- DEFINE TAG
- DEFINE TOKEN
- DELETE ADJUSTMENT
- DELETE ALIAS
- DELETE COMMAND
- DELETE KEY
- DELETE KEYWORDS
- DELETE LANGUAGE
- DELETE PACKAGE
- DELETE PARAMETER
- DELETE PLACEHOLDER
- DELETE ROUTINE
- DELETE TAG
- DELETE TOKEN
- DO
- END DEFINE
- EXTEND
- EXTRACT ADJUSTMENT
- EXTRACT ALIAS
- EXTRACT KEYWORDS
- EXTRACT LANGUAGE
- EXTRACT PACKAGE
- EXTRACT PARAMETER
- EXTRACT PLACEHOLDER
- EXTRACT ROUTINE
- EXTRACT TAG
- EXTRACT TOKEN
- MODIFY LANGUAGE
- SAVE ENVIRONMENT
- SAVE SECTION
- SET MODE
- SET SEARCH

4.15 Help and Status Commands

HELP	SHOW MODULE
SHOW ADJUSTMENT	SHOW PACKAGE
SHOW ALIAS	SHOW PARAMETER
SHOW BUFFER	SHOW PLACEHOLDER
SHOW CMS	SHOW QUERY
SHOW COMMAND	SHOW ROUTINE
SHOW DEFAULT_DIRECTORY	SHOW SCREEN
SHOW DIRECTORY	SHOW SEARCH
SHOW KEY	SHOW SOURCE_DIRECTORY
SHOW KEYWORDS	SHOW SUMMARY
SHOW LANGUAGE	SHOW TAG
SHOW LIBRARY	SHOW TOKEN
SHOW MARK	SHOW VERSION
SHOW MODE	WHAT LINE

4.16 CMS Commands

CMS	SET CMS
REPLACE	UNRESERVE
RESERVE	

Command Descriptions

This section describes LSE and SCA commands in alphabetical order. To aid in differentiating these commands, the following notations appear under the command name:

Notation	Explanation
No notation	LSE standalone commands.
SCA Command	SCA standalone commands. These commands are valid any time that you are using SCA, whether or not you are using LSE.
SCA Required	LSE commands that are valid only if you are using SCA with LSE.

In describing DECwindows menu equivalents for commands, the following terms are used:

Term	Description of action
Button	To activate, press MB1 on item.
Pop-up menu	To activate, press MB2 on first path item; follow path while holding down MB2.
Pull-down menu	To activate, press MB1 on first path item; follow path while holding down MB1.

NOTE

The Language-Sensitive Editor follows the quoting rules of the DIGITAL Command Language (DCL). All references to quoted strings mean that LSE expects double quotation marks (").

In the command descriptions that follow, the defaults for qualifiers are indicated by (D).

ALIGN

Aligns comments within the current selected range without performing a fill operation.

Format

ALIGN

Qualifiers	Defaults
/COMMENT_COLUMN=CONTEXT_DEPENDENT	/COMMENT_COLUMN= CONTEXT_DEPENDENT
/COMMENT_COLUMN=number	/COMMENT_COLUMN= CONTEXT_DEPENDENT

Qualifier

/COMMENT_COLUMN=CONTEXT_DEPENDENT (D)
/COMMENT_COLUMN=number
The /COMMENT_COLUMN=CONTEXT_DEPENDENT qualifier specifies that the comment column should be determined from the context. LSE finds the first trailing comment in the range, uses the starting position of that comment as the comment column, and adjusts all subsequent comments to conform with the first. This is the default.

The /COMMENT_COLUMN=number qualifier specifies the column in which to align the comments. All trailing comments in the range are aligned with the specified column number, which must be an integer ranging from 1 to 131.

ALIGN

Description

The ALIGN command aligns all trailing comments with a particular column. The column in which you position the comments can be either explicitly specified (using the /COMMENT_COLUMN=number qualifier) or based upon context.

This command operates on each line in the range, in sequence. For each line, LSE checks to see whether the line has a trailing comment. If not, it proceeds to the next line.

If there is a trailing comment, then LSE either inserts or deletes spaces or tabs as necessary to get the comment to align. If there is no room for the comment on the line (that is, if the noncommented text extends beyond the comment column), then the comment is aligned one space after the end of the noncommented text.

DECwindows Interface Equivalent

Pull-down menu: Format --> Align

Related Commands

FILL

Example

The following is a sample of commented code:

```
IF (col >= R_Margin) THEN (* This is the start of a *)
      BEGIN            (* bracketed comment sequence that *)
      VAR x: INTEGER;      (* extends over several lines *)
```

Issuing the ALIGN command causes LSE to rearrange the text as follows:

```
IF (col >= R_Margin) THEN (* This is the start of a          *)
      BEGIN               (* bracketed comment sequence that *)
      VAR x: INTEGER;     (* extends over several lines      *)
```

ANALYZE

Creates an analysis data file that describes a source file.

Format

ANALYZE *file-spec[, . . .]*

Qualifiers	**Defaults**
/[NO]DESIGN[=design-option]	/NODESIGN
/LANGUAGE=language	
/[NO]LOG	/LOG
/OUTPUT[=file-spec]	/OUTPUT=file-name.ANA

Qualifiers

/DESIGN[=design-option]
/NODESIGN (D)
Indicates that the source file should be processed as a program design language. The design options are as follows:

COMMENTS	The ANALYZE command looks inside comments for design information. Information about comments is included in the analysis data file. Any errors detected are reported
NOCOMMENTS	The ANALYZE command ignores comments.
PLACEHOLDERS	The ANALYZE command treats LSE placeholders as valid syntax. Placeholders are reported in the analysis data file.
NOPLACEHOLDERS	The ANALYZE command does not report placeholders in the .ANA file. It does not report errors if placeholders are encountered.

If you specify the /DESIGN qualifier, the default is /DESIGN=(COMMENTS,PLACEHOLDERS). If you do not specify this qualifier, the default is /NODESIGN.

/LANGUAGE=language
Specifies the language of the source file. By default, the language is that which is specified by the file type of the source file.

/LOG (D)
/NOLOG
Indicates whether each analyzed file is reported.

/OUTPUT[=file-spec]
/OUTPUT=file-name.ANA (D)
Specifies the analysis data file to be created. The default is /OUTPUT= filename.ANA, where *file-name* is the name of the first source file specified as the parameter to this command.

Parameter

file-spec[, . . .]
Specifies the files to be analyzed. You can use wildcards with the *file-spec* parameter. Within LSE, the current buffer is analyzed by default.

Description

The ANALYZE command creates an analysis data file to describe a source file. The analysis data files produced by this command contain a minimal description of the source file. These files describe the source file primarily as a set of references to unbound names.

With the ANALYZE command, you can use SCA with languages not directly supported by SCA. Do not use this command with those languages that do support SCA.

The ANALYZE command understands the language-specific rules for forming names (identifiers), comments, quoted strings, and placeholders. It assumes that tokens are reserved words, and does not include them in the analysis data file. It processes placeholders and comments depending on the setting of the /DESIGN qualifier.

You must have a language defined in an environment file to use the ANALYZE command with that language. Based on the description of the language in that file, this command analyzes the source file.

The ANALYZE command uses the LSE environment files to determine the appropriate language based on the file type, or uses the language specified with the /LANGUAGE qualifier. It uses the same logical names as LSE, (LSE$ENVIRONMENT and LSE$SYSTEM_ENVIRONMENT) to access the environment files.

For information about defining your own language, see the chapter on defining LSE templates in the *VAX Language-Sensitive Editor and VAX Source Code Analyzer User Manual.*

The REPORT command requires that LSE be installed even if you are using this command from the SCA command line.

Related Commands

DEFINE LANGUAGE
LOAD

EXAMPLES

1.
```
LSE> ANALYZE/LANGUAGE=EXAMPLE PROG1.EXAMPLE
```

Produces an analysis data file that describes an EXAMPLE language souce file.

2.
```
LSE> ANALYZE/DESIGN=(NOPLACEHOLDERS) PROG2.SDML
```

Produces an analysis data file and indicates that the source file should be processed as a program design language. Placeholders are not reported in the .ANA file. By default, information about comments are reported. The language is SDML, as determined by the file type of the source file.

ATTACH

Allows you to switch control of your terminal to another process.

NOTE

This function is not available in DECwindows; any attempt to invoke it incurs an error.

Format

ATTACH *[subprocess-name]*

Parameter

subprocess-name
Specifies the name of the process to which you want to connect. If you do not specify a process name, LSE connects you to the parent process.

Description

The ATTACH command switches control of your terminal to another process, just as the DCL ATTACH command does at the dollar sign ($) prompt. To return to LSE from another process, you use the DCL ATTACH command; you use the LOGOUT command to return to LSE only from a subprocess.

Related Commands

SPAWN

Example

```
LSE> ATTACH SMITH_1
```

Switches control to the process SMITH_1.

CALL

Calls the indicated VAX Text Processing Utility (VAXTPU) procedure.

Format

CALL *VAXTPU-procedure-name [additional-parameters]*

Parameters

VAXTPU-procedure-name
Indicates the name of the VAXTPU procedure you want to call.

additional-parameters
Contains information to be passed to the procedure as a single string. The called procedure must then parse and interpret this string.

Description

The CALL command, in combination with the DEFINE COMMAND command, provides a means for defining new commands implemented in the VAXTPU language. Because the additional parameters are passed to the called procedure without being parsed, these commands have a flexible syntax.

Related Commands

DEFINE COMMAND
DO/TPU

Example

The following VAXTPU procedure issues a DIRECTORY command from within LSE:

```
PROCEDURE dir (dir_params)
    ! Description:
    !   Issues a DCL DIRECTORY command in a subprocess.  The output is
    !   written to the DIRECTORY buffer.  The DIRECTORY buffer is
    !   mapped to the current window.
    !
    ! Parameter:
    !   dir_params - a string beginning with "$".  The text following
    !       the "$" contains parameters and qualifiers to be passed to
    !       the DIRECTORY command.
    !       The "$" is used to provide a parameter for the call to
    !       this procedure when no parameters for the DIRECTORY
    !       command were specified.
    !
    LOCAL dir_process, cmd;
    IF GET_INFO(dir_buffer, "TYPE") <> BUFFER THEN
        dir_buffer := CREATE_BUFFER("DIRECTORY");
        SET(NO_WRITE,dir_buffer);
    ENDIF;
    erase(dir_buffer);

    ! Build the DIRECTORY command, picking up parameters that were
    ! passed in.
    cmd := 'DIRECTORY '+SUBSTR(dir_params,2,LENGTH(dir_params)-1);
    ! Create a subprocess and execute the command.
    dir_process := CREATE_PROCESS (dir_buffer, cmd);
    lse$do_command('GOTO BUFFER DIRECTORY');

    DELETE (dir_process);
ENDPROCEDURE
```

To define this procedure, you put it in a buffer and compile it using the DO/TPU command. To use the procedure, define a command named DIR, as follows:

```
LSE> DEFINE COMMAND DIR "CALL DIR $"
```

To get a directory listing, enter your newly defined DIR command, as follows:

```
LSE> DIR/SIZE/DATE
```

Note the use of the dollar sign ($) to cause the CALL command to always invoke the procedure named DIR with a parameter, even if you specify nothing else on the command line of the command DIR. The dollar sign also prevents qualifiers on the command DIR from being interpreted as an attempt to place qualifiers on the VAXTPU *procedure-name* parameter named DIR.

CANCEL MARK

Cancels the indicated marker set by a SET MARK command.

Format

CANCEL MARK *[marker-name]*

Parameter

marker-name
Names the marker to be canceled; a wildcard marker name is allowed. If you do not specify a name for the marker, LSE cancels any marker at the current cursor position.

Description

The CANCEL MARK command causes LSE to remove the indicated marker from the text and to delete the marker name.

Related Commands

SET MARK

Example

```
LSE> CANCEL MARK 1
```

Deletes the marker named *1* from the inventory of markers in your current buffer.

CANCEL SELECT_MARK

Cancels the selected range of the SET SELECT_MARK command.

Format

CANCEL SELECT_MARK

Description

The CANCEL SELECT_MARK command cancels the effect of the SET SELECT_MARK command. If a block or range of text is displayed in reverse video, the CANCEL SELECT_MARK command returns the text to its normal display.

Keypad Equivalent

Key	**Keypad Mode**
PF1-Keypad period (.) RESET	EDT LK201, EDT VT100, EVE LK201
PF1-E4 SELECT	EVE LK201

Related Commands

SET SELECT_MARK

CAPITALIZE WORD

Capitalizes the first letter of the current word, or words, in a selected range.

Format

CAPITALIZE WORD

Description

The CAPITALIZE WORD command capitalizes the first letter of the word following the cursor, or the word that the cursor is on. If a selected range is active, all the words within that range are capitalized.

If a word is already in uppercase letters, the command changes all but the first letter to lowercase. The cursor then moves to the first letter of the word following the target word or selected range.

DECwindows Interface Equivalent

Pull-down menu: Format --> Capitalize

Related Commands

CHANGE CASE
LOWERCASE WORD
UPPERCASE WORD

CENTER LINE

Centers the current line between the left and right margins.

Format

CENTER LINE

Description

The CENTER LINE command centers text on the line that the cursor is on. You may place the cursor anywhere on the line to be centered.

DECwindows Interface Equivalent

Pull-down menu: Format --> Center Line

CHANGE CASE

Changes the case of a letter, or letters, in a selected range.

Format

CHANGE CASE

Description

The CHANGE CASE command changes the case of letters.

If you have selected a range of text by using the SET SELECT_MARK command, the case of each letter in the selected range changes.

Keypad Equivalent

Key	Keypad Mode
PF1-KP1 [CHNGCASE]	EDT LK201, EDT VT100, EVE LK201
None	EVE VT100

Related Commands

CAPITALIZE WORD
LOWERCASE WORD
UPPERCASE WORD

CHANGE DIRECTION

Changes the current direction of the current buffer between forward and reverse.

Format

CHANGE DIRECTION

Description

The CHANGE DIRECTION command changes the current direction attribute of the current buffer. The buffer's status line indicates whether the current direction is forward or reverse. The direction affects the operation of such commands as GOTO, ERASE, SEARCH, SUBSTITUTE, CHANGE INDENTATION, and CHANGE CASE.

With the DECwindows interface, you can switch directions by moving the mouse cursor to Forward or Reverse on the status line and then pressing MB1.

Keypad Equivalent

Key	Keypad Mode
F11 [FORWARD REVERSE]	EVE LK201, EDT LK201
PF3 [FORWARD REVERSE]	EVE VT100
None	EDT VT100

DECwindows Interface Equivalent

Button: Buffer status $\left\{ \begin{array}{l} \text{Forward} \\ \text{Reverse} \end{array} \right\}$

Related Commands

SET FORWARD
SET REVERSE

CHANGE INDENTATION

Adds or removes leading blanks and tabs from lines.

Format

CHANGE INDENTATION

Qualifiers	Defaults
/CURRENT	/CURRENT
/FORWARD	/CURRENT
/[NO]HOLD	/HOLD
/REVERSE	/CURRENT

Qualifiers

/CURRENT (D)
Specifies the current direction for the change in indentation.

/FORWARD
Specifies the forward direction for the change in indentation.

/HOLD (D)
/NOHOLD
Specifies whether or not the selected range is canceled by this command. You use the /HOLD qualifier to keep the selected range active so that you may repeat this command to make incremental changes in indentation.

/REVERSE
Specifies the reverse direction for the change in indentation.

Description

The CHANGE INDENTATION command adds or removes blanks and tabs from the line that the cursor is on, and sets the new indentation of the current line as the current indentation level.

If you select a range of text by using the SET SELECT_MARK command, the CHANGE INDENTATION command adds or removes blanks and tabs from each line of text in the selected range.

Keypad Equivalent

CHANGE INDENTATION/FORWARD

Key	Keypad Mode
PF1-→ IND FWD	EDT LK201, EDT VT100
PF1-]	All

CHANGE INDENTATION/REVERSE

Key	Keypad Mode
PF1-← IND REV	EDT LK201, EDT VT100
PF1-[	All

DECwindows Interface Equivalent

$$\text{CHANGE INDENTATION} \left\{ \begin{array}{l} \text{/FORWARD} \\ \text{/REVERSE} \end{array} \right\}$$

Pull-down menu: Format --> Indentation . . .

Related Commands

ENTER TAB
SET INDENTATION
TAB
UNTAB

CHANGE TEXT_ENTRY_MODE

Switches the mode of text-entry in the current buffer between insert and overstrike.

Format

CHANGE TEXT_ENTRY_MODE

Description

The CHANGE TEXT_ENTRY_MODE command switches the mode of the current buffer between insert and overstrike. The status line displays the current text-entry mode.

Keypad Equivalent

Key	**Keypad Mode**
F14 INSERT OVERSTR	EDT LK201, EVE LK201
CTRL/A	All
ENTER INSERT OVERSTR	EVE VT100

DECwindows Interface Equivalent

$$\textbf{Button}\text{: Buffer status line} \left\{ \begin{array}{l} \text{Insert} \\ \text{Overstrike} \\ \text{Nomodify} \end{array} \right\}$$

Related Commands

SET INSERT
SET OVERSTRIKE

CHANGE WINDOW_MODE

Switches between reducing and increasing the number of windows displayed on the screen.

Format

CHANGE WINDOW_MODE

Description

The CHANGE WINDOW_MODE command changes the number of windows displayed on the screen. If the screen has one window, this command creates a second window. If the screen has two or more windows, the CHANGE WINDOW_MODE command reduces the screen display to a single window containing the current buffer.

Keypad Equivalent

Key	Keypad Mode
PF1-equal sign (=)	All

Related Commands

ONE WINDOW
SET SCREEN WINDOW

CHECK LANGUAGE

Analyzes the definitions associated with a language and reports errors.

Format

CHECK LANGUAGE *language-name*

Qualifiers	Defaults
/DEFINITIONS	/DEFINITIONS
/HELP_INTERFACE	/DEFINITIONS

Qualifiers

/DEFINITIONS (D)
Specifies that the CHECK LANGUAGE command report the following:

- Undefined tokens
- Undefined placeholders
- Unreferenced placeholders
- Package routines with the same name as placeholders
- Package parameters with the same name as tokens
- Parameters defined with the same name in multiple packages
- Routines defined with the same name in multiple packages
- Invalid topic strings

/HELP_INTERFACE
Specifies that the CHECK LANGUAGE command report invalid topic strings.

Parameter

language-name
Specifies the name of the language whose definitions are to be checked. Wildcards are not permitted.

Restrictions

The /DEFINITIONS and /HELP_INTERFACE qualifiers are mutually exclusive.

Description

The CHECK LANGUAGE command analyzes the definitions associated with a language. This command detects and reports the following:

- Undefined tokens—An undefined token has not been defined by a DEFINE TOKEN command but appears in a menu placeholder body.
- Undefined placeholders—An undefined placeholder has not been defined by a DEFINE PLACEHOLDER command. It appears in the body of a token or in the body of a nonterminal or menu placeholder; or it appears as the value of a /PLACEHOLDER qualifier on a DEFINE TOKEN or DEFINE PLACEHOLDER command.
- Unreferenced placeholders—An unreferenced placeholder has been defined using a DEFINE PLACEHOLDER command. It does not appear in the body of any token or in the body of any nonterminal or menu placeholder; and it is not used as the value of a /PLACEHOLDER qualifier on a DEFINE TOKEN or DEFINE PLACEHOLDER command.
- Package routines with the same name as placeholders—A token name that is the same as the routine name in a package associated with the language prevents LSE from accessing the template for the routine.
- Package parameters with the same name as tokens —A placeholder name that is the same as the parameter name in a package associated with the language prevents the VAXTPU procedure associated with the parameters for that package from being called to properly define the

placeholder(s) for the parameter. It may cause incorrect behavior and erroneous messages.

- Parameters defined with the same name in multiple packages—A parameter name that is defined in multiple packages associated with the language may cause the wrong VAXTPU procedure to be called for the parameter. It may cause incorrect behavior and erroneous messages. A parameter is not reported as defined in multiple packages if the packages have been defined with the same VAXTPU procedure for parameter expansion, that is, the same value on the /PARAMETER_EXPAND qualifier on DEFINE PACKAGE commands.
- Routines defined with the same name in multiple packages—A routine that is defined in multiple packages associated with the language prevents LSE from accessing some of the routine templates, since it will expand only the first definition for a routine that it encounters.
- Invalid topic strings—Topic strings are specified as values on the /TOPIC qualifier on the DEFINE TOKEN and DEFINE PLACEHOLDER commands. A topic string is invalid if there is no corresponding HELP text in the HELP library for the language.

If LSE detects any of the above conditions, those conditions are reported in the $CHECK_LANGUAGE buffer, which is displayed in an editing window. You can use the WRITE command to write the contents of this buffer to a file. If LSE does not detect any of the conditions listed above, a success message is displayed.

Example

```
$ LSEDIT /NODISPLAY /NOCURRENT_FILE /INITIALIZATION=SYS$INPUT:
CHECK LANGUAGE/HELP_INTERFACE my_language
WRITE /BUFFER=$CHECK_LANGUAGE CHECK_LANGUAGE.LIS
QUIT
$
```

The size and structure (key depth) of the HELP library and the number of tokens and placeholders that have /TOPIC qualifiers determine the amount of time required to check the HELP library for the language. This checking can take a significant amount of time. It may be more convenient to check the HELP library for a language from a batch procedure.

CLOSE BUFFER

Writes and deletes the current buffer.

Format

CLOSE BUFFER

Description

The CLOSE BUFFER command writes the buffer contingent on buffer attributes and status and then deletes the buffer. If the buffer has the WRITE attribute, and you have modified the contents of the buffer since they were last written, LSE first writes the contents of the buffer to its associated file. If a file is not associated with the buffer, LSE prompts you for a file name.

DECwindows Interface Equivalent

Pop-up menu: User buffer --> Close
Pull-down menu: File --> Close

Related Commands

WRITE

CMS

Invokes VAX DEC/Code Management System (CMS) to enable any valid CMS command to execute from within LSE.

Format

CMS *[cms-command]*

Parameter

cms-command
Specifies any valid specification for a CMS command, including valid qualifiers.

Description

The CMS command invokes CMS from within LSE to let you issue any valid CMS command. To use this command, you must have CMS installed on your system.

Related Commands

GOTO FILE
GOTO SOURCE
READ
REPLACE
RESERVE
SET CMS
UNRESERVE

Examples

1.
```
LSE> CMS SET LIBRARY DISK$:[USER.CMSLIB]
```

Sets the specified library as your current CMS library. Thereafter, LSE file-manipulation commands, such as GOTO FILE, GOTO SOURCE, and READ, access that library when you issue a SET CMS command.

2.
```
LSE> CMS SHOW RESERVATIONS
```

Reports on the reservation history of all elements in the library set as your current CMS library.

COLLAPSE

Compresses text at the current cursor position.

Format

COLLAPSE

Qualifier	**Default**
/DEPTH=n	/DEPTH=1

Qualifier

/DEPTH=n
/DEPTH=1 (D)
Compresses the text at the current cursor position up *n* levels. If you specify the value ALL, this qualifier compresses the text at the cursor position as much as possible.

Note that when you use the COLLAPSE command in query buffers, this command does not support the /DEPTH qualifier.

Description

The COLLAPSE command displays an overview of the text at the current cursor position. Low-level detail lines are replaced by a single overview line. The cursor position is recorded before the text is collapsed for use with future EXPAND commands.

The editor determines the relative level of detail of a line by comparing the indentation of the line with the indentation of other lines. The editor's treatment of the indentation of a line is influenced by indentation adjustment definitions. For more information, see the DEFINE ADJUSTMENT command.

In an SCA query buffer, if the cursor is positioned on a symbol that has been expanded, or on an occurrence within an expansion, the COLLAPSE command causes the occurrences to disappear.

Keypad Equivalent

Key	Keypad Mode
CTRL \	ALL

DECwindows Interface Equivalent

COLLAPSE
Pop-up menu: Query buffer --> Collapse
Pull-down menu: View --> Collapse

COLLAPSE/DEPTH=ALL
Pull-down menu: View --> Collapse All

Related Commands

DEFINE ADJUSTMENT
DEFINE LANGUAGE/OVERVIEW_OPTIONS
EXPAND
FOCUS
MODIFY LANGUAGE
SET NOOVERVIEW
SET OVERVIEW
VIEW SOURCE

COMPILE

Lets you compile the contents of a buffer without leaving LSE.

Format

COMPILE *[command-string]*

Qualifier	**Default**
/[NO]REVIEW	/NOREVIEW

Qualifier

/REVIEW
/NOREVIEW (D)
Tells LSE whether or not to wait for the spawned subprocess to complete and then to automatically review any errors reported by the compiler. If you do not specify this qualifier when compiling, you can use the REVIEW command to display any errors after compilation.

By default, the COMPILE command completes as soon as compilation starts. Specifying the /REVIEW qualifier causes the review process to occur as soon as compilation completes.

Parameter

command-string
Specifies the DCL command line to be executed. If you do not specify a command string, LSE uses the command string specified in the definition of the language associated with the current buffer (see the DEFINE LANGUAGE/COMPILE_COMMAND command).

If you specify a dollar sign ($) as the first argument on the COMPILE command, LSE replaces the dollar sign with the default COMPILE command. With this feature, you can append file specifications or command qualifiers to the default COMPILE command without having to type the entire command yourself.

If the command string or the string specified on the /COMPILE_COMMAND qualifier contains LSE$FILE, then LSE forms the command used to compile the buffer by substituting, for LSE$FILE, the file specification that corresponds to the buffer. With this feature, you can insert text on the command line immediately after the file specification and before the /DIAGNOSTICS qualifier. If the COMPILE command does not contain LSE$FILE, then LSE appends the file specification to the string specified on the qualifier.

Description

The COMPILE command compiles the contents of a buffer without leaving LSE. When you issue this command, LSE writes the current buffer and other buffers associated with the current language back to their files if they have been modified since they were last written. A buffer is not written if it is designated READ_ONLY.

LSE then forms a DCL command line by appending the file specification of the current buffer to the command string given on the COMPILE command.

If the current buffer's language has diagnostic capabilities (see the DEFINE LANGUAGE/CAPABILITIES command), LSE appends the /DIAGNOSTICS qualifier to the DCL command it forms, as follows:

```
/DIAGNOSTICS=current-device:[current-directory]filespec.DIA
```

LSE then spawns a subprocess to execute the DCL command line.

If you specified the /REVIEW qualifier on the COMPILE command, LSE waits for the subprocess to finish executing the DCL command. Otherwise, the COMPILE command completes as soon as the subprocess begins executing the DCL command.

When the subprocess completes, LSE displays a message in the message buffer. If you specified the /REVIEW qualifier, LSE enters review mode and reviews any compilation errors that occurred.

DECwindows Interface Equivalent

Pull-down menu: File --> Compile

Examples

1.
```
LSE>  COMPILE $/DEBUG
```

Compiles the contents of the current buffer. If that buffer is named PROG.FOR, and the current directory specification is USER$:[SMITH], then the following DCL command executes:

```
$ FORTRAN/DEBUG PROG.FOR/DIAGNOSTICS=USER$:[SMITH]PROG.DIA
```

With this command, suppose you had previously specified the following:

```
DEFINE LANGUAGE/COMPILE_COMMAND="FORTRAN 'LSE$FILE'+XXX"
```

Then the DCL command that executes would be as follows:

```
$ FORTRAN PROG.FOR+XXX/DEBUG /DIAGNOSTICS=USER$:[SMITH]PROG.DIA
```

2.
```
LSE>  COMPILE FORTRAN 'LSE$FILE'+YYY
```

Compiles the contents of the current buffer. If that buffer is named PROG.FOR, and the current directory specification is USER$:[SMITH], then the following DCL command executes:

```
$ FORTRAN PROG.FOR+YYY /DIAGNOSTICS=USER$:[SMITH]PROG.DIA
```

Note that the /DIAGNOSTICS qualifier is appended to the FORTRAN command if you specified that qualifier in the DEFINE LANGUAGE/CAPABILITIES command.

3.
```
$ OPEN/WRITE X BCOMP.COM
$ WRITE X "$ FORTRAN ",P1," ",P2
$ CLOSE X
$ SUBMIT/NOPRINT/DELETE BCOMP
$ SYNCHRONIZE BCOMP
```

This example is a command procedure, named FORTBATCH.COM, that you could submit as a batch job to compile a FORTRAN program.

To submit this as a batch job to the FORTRAN compiler, enter the following command:

```
LSE>  COMPILE @FORTBATCH
```

If the current buffer is A.FOR and contains a FORTRAN program, LSE writes A.FOR to the disk and spawns a subprocess to execute the following DCL command:

```
$ @FORTBATCH A.FOR;2 /DIAGNOSTICS=DISK$:[USER]A.DIA
```

This causes the FORTBATCH procedure to create the file BCOMP.COM, which contains the following DCL compilation command:

```
$ FORTRAN A.FOR;2 /DIAGNOSTICS=DISK$:[USER]A.DIA
```

The FORTBATCH procedure then submits BCOMP.COM to run in batch.

The DCL command SYNCHRONIZE (on the final line of the sample command procedure) causes the subprocess to wait until the batch job completes before it returns control to LSE. This is essential if you are specifying the COMPILE/REVIEW command. LSE considers the compilation to be completed when the subprocess finishes executing. If you do not specify the SYNCHRONIZE command upon completion of a batch job, you cannot use the COMPILE/REVIEW command. However, you can use LSE's REVIEW command to enter Review mode after the batch job finishes the compilation.

CONTINUE

Ends command entry and returns control to keypad-mode editing.

Format

CONTINUE

Description

With the CONTINUE command, you can return to keypad editing from the command prompt. You can also press CTRL/Z at the LSE> prompt to return to keypad editing.

Keypad Equivalent

CTRL/Z, at LSE> prompt

Related Commands

DO

CONVERT LIBRARY

Converts the specified library from Version 1.*n* format to Version 2.0 format.

Format

CONVERT LIBRARY *directory-spec1 [directory-spec2]*

Parameters

directory-spec1
Specifies the directory specification of the Version 1.*n* library to be converted.

directory-spec2
Specifies the directory in which the Version 2.0 library is to be created. If this parameter is omitted, the new library is created in the directory specified by *directory-spec1*, and the old library is deleted. If this parameter is specified and is different from *directory-spec1*, the old library is not deleted.

Description

The CONVERT LIBRARY command converts a Version 1.*n* library to a format compatible with Version 2.0. Because Version 2.0 libraries can contain much more information than Version 1.*n* libraries, it is recommended that you recompile and load new libraries rather than convert libraries, if possible.

Related Commands

CREATE LIBRARY
LOAD

Example

```
LSE>  SCA CONVERT LIBRARY SCA$:[USER.V1LIB] SCA$:[USER.V2LIB]
```

Uses the existing library [USER.V1LIB] to create a new Version 2.0 library named [USER.V2LIB].

CREATE LIBRARY

Allocates and initializes VMS library files in a specified directory. The new library then becomes an active SCA library.

Format

CREATE LIBRARY *directory-spec[, . . .]*

Qualifiers	**Defaults**
/AFTER=[library-spec]	
/BEFORE=[library-spec]	
/[NO]LOG	/LOG
/MODULES=module-count	/MODULES=25
/[NO]REPLACE	/NOREPLACE
/SIZE=block-count	/SIZE=1000

Qualifiers

/AFTER=library-spec
Instructs SCA to insert the new library or libraries into the list of active SCA libraries following the library you specify as the value of the qualifier. If you do not specify a value, SCA adds the library or libraries to the end of the list.

/BEFORE=library-spec
Instructs SCA to insert the new library or libraries into the list of active SCA libraries in front of the library you specify as the value of the qualifier. If you do not specify a value, SCA adds the library or libraries to the beginning of the list.

/LOG (D)
/NOLOG
Indicates whether SCA reports the successful creation of a library and the resulting list of active libraries.

/MODULES=module-count
/MODULES=25 (D)
Specifies an estimated number of modules in the library.

/REPLACE
/NOREPLACE (D)
Indicates whether LSE replaces an existing library with a new (empty) library.

/SIZE=block-count
/SIZE=1000 (D)
Specifies an estimated size for a library.

Parameter

directory-spec[, . . .]
Specifies one or more directories in which library files are to be allocated and initialized.

Description

The CREATE LIBRARY command initializes a library and defines it as the active library in your current SCA session. When you subsequently invoke SCA, it uses the logical name SCA$LIBRARY to reestablish the active library list.

Related Commands

CONVERT LIBRARY
LOAD
SET LIBRARY

Examples

1.
   ```
   $ SCA CREATE LIBRARY SCA$:[USER.SCA]/REPLACE
   ```

   Initializes a library in the named directory. SCA replaces the existing library with empty library files.

2.
   ```
   $ SCA CREATE LIBRARY TOP
   ```

   Initializes a library in the directory defined by logical name TOP.

   For additional examples, see the section about creating a library in the *VAX Language-Sensitive Editor and VAX Source Code Analyzer User Manual.*

CUT

Moves or copies the selected range to the indicated buffer.

Format

CUT

Qualifiers	**Defaults**
/[NO]APPEND	/NOAPPEND
/BUFFER=buffer-name	/BUFFER=$PASTE (D)
/CLIPBOARD	See Description
/[NO]ERASE	/ERASE
/REPLACE	
/SUBSTITUTE	

Qualifiers

/APPEND
/NOAPPEND (D)
Indicates whether the moved text should be appended to the current contents of the receiving buffer or should replace the current contents of the receiving buffer.

/BUFFER=buffer-name
/BUFFER=$PASTE (D)
Specifies the buffer to receive the text being moved. If the /REPLACE or /SUBSTITUTE qualifier is specified, then the indicated buffer supplies text to replace text being erased from the current buffer.

/CLIPBOARD
Specifies that the DECwindows clipboard should be used to receive the text being moved, instead of a buffer. The /CLIPBOARD and /BUFFER qualifiers are mutually exclusive.

/ERASE (D)
/NOERASE
Specifies whether or not the moved text should be deleted from the current buffer. LSE ignores this qualifier if the current buffer is not modifiable.

/REPLACE
Erases the selected text and replaces it with the contents of the indicated buffer.

/SUBSTITUTE
Erases the search string, replaces it with the contents of the buffer indicated in the /BUFFER qualifier, and finds the next occurrence of the string. To use this qualifier, do the following:

1. Issue the SET SELECT_MARK command (press the SELECT key) at the command prompt.
2. Type the new text in the buffer.
3. Issue the CUT command (press the CUT or REMOVE key) at the command prompt. This places the text in the indicated buffer.
4. Issue the SEARCH command (press the FIND key) at the command prompt, followed by the text you want to search for and replace.
5. Press the ENTER key.
6. Issue the CUT/SUBSTITUTE command (press the SUBS key) at the command prompt.

Subsequently, each time you issue the CUT/SUBSTITUTE command, LSE makes one substitution and finds the next occurrence of the search string.

Description

The CUT command removes or copies text within the selected range and moves it into a designated buffer or default location (the DECwindows Clipboard or character cell $PASTE buffer). The selected range is the text between the select marker (see the SET SELECT_MARK command) and the current cursor position. If no select marker has been set, and the cursor is positioned on the current search string, then that string is moved to the buffer.

The /REPLACE and /SUBSTITUTE qualifiers are mutually exclusive; also, these qualifiers may not be used in conjunction with the /APPEND and /ERASE qualifiers.

For users of the DECwindows interface, the default setting is /CLIPBOARD; otherwise, the default is /BUFFER=$PASTE.

Keypad Equivalent

CUT

Key	Keypad Mode
KP6 CUT	EVE LK201, EDT LK201, EDT VT100
E3 REMOVE	EDT LK201, EVE LK201
KP8 REMOVE	EVE VT100

CUT/APPEND

Key	Keypad Mode
KP9 APPEND	EDT LK201, EDT VT100, EVE LK201
None	EVE VT100

CUT/NOERASE

Key	Keypad Mode
PF1-/E3 COPY	EDT LK201, EVE LK201

CUT/REPLACE

Key	Keypad Mode
PF1-KP9 REPLACE	EDT LK201, EDT VT100, EVE LK201
None	EVE VT100

CUT/SUBSTITUTE

Key	Keypad Mode
PF1-Enter SUBS	EDT LK201, EDT VT100, EVE LK201
None	EVE VT100

DECwindows Interface Equivalent

CUT/CLIPBOARD
Pop-up menu: User buffer --> Cut
Pull-down menu: Edit --> Cut

CUT/NOERASE/CLIPBOARD
Pop-up menu: User buffer --> Copy
Pull-down menu: Edit --> Copy

Related Commands

PASTE
SET SELECT_MARK
SUBSTITUTE

Example

```
LSE>  CUT/BUFFER=TEMP.TXT
```

Places the text being moved in the buffer TEMP.TXT.

DCL

Executes a DCL command from within your editing session.

Format

DCL *dcl-command*

Parameter

dcl-command
Specifies the DCL command to be executed. If you do not specify a command, LSE prompts for one. Pressing CTRL/Z at the prompt cancels the operation.

LSE splits the window to show the DCL buffer. You can edit the DCL buffer to move the output from the DCL command into another buffer. You can use the ONE WINDOW command to remove the DCL window.

Description

The LSE command DCL executes a DCL command from within your editing session. LSE spawns a subprocess for the DCL command you specify and creates a buffer named DCL to contain the output from the command.

Example

```
LSE>  DCL DIRECTORY *.TXT
```

Splits the screen and displays the DCL command DIRECTORY and its output (the directory listing) in the second window. The cursor remains in the first window.

DEFINE ADJUSTMENT

Defines the behavior of the LSE viewing commands on individual lines of a source file.

Format

DEFINE ADJUSTMENT *adjustment-name [pattern]*

Qualifiers	**Defaults**
/[NO]COMPRESS	/COMPRESS
/[NO]COUNT	/COUNT
/CURRENT=number	/CURRENT=0
/[NO]INHERIT=inherit-keyword	/NOINHERIT
/LANGUAGE=language-name	Current buffer language
/[NO]OVERVIEW	/OVERVIEW
/[NO]PREFIX=(indentation-value, adjustment-value)	/NOPREFIX
/SUBSEQUENT=number	/SUBSEQUENT=0
/[NO]UNIT	/NOUNIT

Qualifiers

/COMPRESS (D)
/NOCOMPRESS
Avoids compressing groups and overrides indentation. If a group of lines begins with a /NOCOMPRESS line, then the group is never compressed.

/COUNT (D)
/NOCOUNT
Controls whether the matching line contributes to the line count for the group. When determining whether to form a group, the line count is compared with the *minimum_lines* value for the language.

See the description for DEFINE LANGUAGE/OVERVIEW_OPTIONS= MINIMUM_LINES.

/CURRENT=number
/CURRENT=0 (D)
Adjusts the indentation of the current line. If a buffer line matches an adjustment defined with the /CURRENT qualifier, then the indentation of the buffer line is adjusted by the number of columns given as the qualifier value. A positive value causes the indentation to be adjusted to the right; a negative value causes the indentation to be adjusted to the left. For example, DEFINE ADJUSTMENT then /CURRENT=1 means "Adjust each line that begins with the word 'then' one column to the right."

See the DEFINE LANGUAGE/OVERVIEW_OPTIONS=TAB_RANGE description.

/INHERIT=inherit-keyword
/NOINHERIT (D)
Specifies that the indentation for the current line is taken from the adjusted indentation of another line.

You can specify one of the following keywords to determine the indentation of the current line:

Keyword	Description
MAXIMUM	The visible indentation for the current line is taken from the adjusted indentation of either the previous line or the next line, whichever is larger.
MINIMUM	The visible indentation for the current line is taken from the adjusted indentation of either the previous line or the next line, whichever is smaller.
NEXT	The visible indentation for the current line is taken from the adjusted indentation of the next line.
PREVIOUS	The visible indentation for the current line is taken from the adjusted indentation of the previous line.

You cannot specify the /INHERIT qualifier with either the /PREFIX qualifier or the /SUBSEQUENT qualifier.

/LANGUAGE=language-name
Specifies the language associated with the indentation adjustment. By default, the new adjustment is associated with the language for the current

buffer. If there is no language associated with the current buffer, then the /LANGUAGE qualifier is required.

/OVERVIEW (D)
/NOOVERVIEW
Controls whether or not the text of the line is used as the overview line. If a line matches an adjustment defined with the /NOOVERVIEW qualifier, then the text of the line is never used as the overview text for compressed lines. Instead, text from a later line is used as the overview text. The /NOOVERVIEW qualifier is used to prevent uninformative text from appearing in overview lines.

/PREFIX=(indentation-value, adjustment-value)
/NOPREFIX (D)
Provides a way to skip a pattern at the beginning of a line to determine indentation or influence adjustment. The /PREFIX qualifier takes the following pair of values:

Indentation-value
Adjustment-value

Indentation-value is one of the following keywords:

- **CURRENT**—Instructs LSE to use the indentation of the first text in the pattern—the beginning of the prefix.
- **FOLLOWING**—Instructs LSE to use the indentation of the text that follows the prefix. If there is no text after the prefix, use the indentation of the prefix.

Adjustment-value is one of the following keywords:

- **CURRENT**—Instructs LSE to use the adjustment qualifier values given on the current definition.
- **FOLLOWING**—Instructs LSE to use the adjustment qualifier values from the definition that matches the text following the prefix. If no text follows the prefix on the current line, LSE uses the qualifier values for a blank line. If /PREFIX has an adjustment value of FOLLOWING, other action qualifiers on the definition are ignored.

The combination (CURRENT,CURRENT) is not useful because it causes both the indentation and the adjustments to be taken from the text at the beginning of the pattern. This is the same as having no prefix at all.

You cannot specify the /PREFIX qualifier with the /INHERIT qualifier.

/SUBSEQUENT=number
/SUBSEQUENT=0 (D)
Adjusts the indentation of lines after the current line. If a buffer line matches an adjustment defined with the /SUBSEQUENT qualifier, then the indentation of all lines after the given one are adjusted by the number of columns given as the qualifier value. A positive value causes the indentation to be adjusted to the right; a negative value causes the indentation to be adjusted to the left.

Use the /SUBSEQUENT qualifier for language constructs that denote nesting and have well-defined endpoints. Use a positive value at the beginning of the construct and a negative value at the end.

You cannot specify the /SUBSEQUENT qualifier with the /INHERIT qualifier.

/UNIT
/NOUNIT (D)
Treats consecutive lines as a single unit. If consecutive lines in the buffer match adjustments defined with the /UNIT qualifier and have the same adjusted indentation, then the sequence of lines is treated as one group, with the first serving as the overview line. Notice that it is not required that all elements of the group match the same adjustment definition; it is only required that the /UNIT qualifier be specified on all the definitions.

Parameters

adjustment-name
Specifies the name of the adjustment being defined.

pattern
Specifes the string that LSE compares against source lines. If no pattern is used, the *adjustment-name* parameter is used. For details about the syntax for pattern strings, see the section about pattern-matching rules in the *VAX Language-Sensitive Editor and VAX Source Code Analyzer User Manual.*

Pattern strings match any string that can be specified directly on the command line. Strings with special characters must be enclosed in quotes (" "). Whether the string is quoted or not, the comparison is case-insensitive. You must use the "$( )" convention to enclose named pattern elements.

Definitions with literal strings take precedence over definitions with predefined patterns.

A list of predefined patterns follows:

- **COLUMN=(first-column[,last-column])**—Limits the column in which the text may start.

  You can specify either the first column or both the first column and the last column. If you specify both the first and last columns, you must enclose the column values in parentheses. If you do not specify the last column, it takes its default from the first column.
- **IDENTIFIER**—Matches a sequence of identifier characters.
- **LINE_END**—Matches the end of a line, optionally preceeded by white space.
- **OPTIONAL_SPACE**—Matches any sequence of spaces and tabs.
- **FORMFEED**—Matches a form-feed character.
- **FORTRAN_COMMENT**—Matches only FORTRAN comment lines.
- **FORTRAN_FUNCTION**— Matches the first line of any FORTRAN function subprogram. That is defined to be any line that matches the following pattern:

  ```
  type [*number] FUNCTION
  ```

  where

  ```
  type :==        BYTE
               |  LOGICAL
               |  INTEGER
               |  REAL
               |  DOUBLE PRECISION
               |  COMPLEX
               |  DOUBLE COMPLEX
               |  CHARACTER

  NUMBER :==     {DIGIT}...
               |  (*)
  ```
- **PREFIX**—The preceding part of the pattern is a prefix.

- **NUMBER**—Matches any sequence of digits. White space may not appear between digits. In the case of a match with both NUMBER and IDENTIFIER, NUMBER takes precedence.

Description

The DEFINE ADJUSTMENT command defines the behavior of the LSE viewing commands on individual lines of a source file. With the DEFINE ADJUSTMENT command, you can modify the behavior of overviews to match your formatting conventions. You can save DEFINE ADJUSTMENT commands in your environment file.

Related Commands

COLLAPSE
DEFINE LANGUAGE/OVERVIEW_OPTIONS
DELETE ADJUSTMENT
EXPAND
EXTRACT ADJUSTMENT
FOCUS
SHOW ADJUSTMENT
VIEW SOURCE

Example

```
LSE> DEFINE ADJUSTMENT then /CURRENT=1
```

Adjusts each line that starts with the word *then* one column to the right.

```
LSE> DEFINE ADJUSTMENT "$(identifier):" /INHERIT=NEXT
```

Specifies that a line starting with any identifier followed by a colon takes the indentation from the following line.

DEFINE ALIAS

Lets you assign an abbreviated sequence of characters to represent a longer string of text. You may then use the EXPAND command to produce the longer string each time the cursor is at the end of the abbreviated sequence.

Format

DEFINE ALIAS *alias-name [value]*

Qualifiers
/INDICATED
/LANGUAGE=language-name

Qualifiers

/INDICATED
Instructs LSE to interpret the contiguous sequence of characters before and after the cursor as the alias (long form) for an alias name (short form) that you supply. To specify which characters are valid in an alias name for the language you are using, issue a DEFINE LANGUAGE command with the /IDENTIFIER_CHARACTERS qualifier.

When you use the /INDICATED qualifier, you must not specify the value parameter.

/LANGUAGE=language-name
Specifies the language associated with the alias. The default is the language for the current buffer.

Parameters

alias-name
Specifies the name to be defined as an alias. The characters in the alias name must be in the /IDENTIFIER_CHARACTERS string in the DEFINE LANGUAGE command.

value

Specifies a quoted string. When you expand the alias, LSE replaces the alias name with the string given by the value parameter. You must not use a value parameter if you specify the /INDICATED qualifier.

Description

With the DEFINE ALIAS command, you can use a shortened name to generate a string of text. You may specify an identifier at the current cursor position as the text you want to generate. Once you have defined an alias name, you can type the alias and then issue the EXPAND command; the text you have assigned to that alias then appears.

Keypad Equivalent

DEFINE ALIAS/INDICATED

Key	Keypad Mode
PF1-CTRL/A	All

Related Commands

DEFINE LANGUAGE
EXPAND

Example

```
LSE>  DEFINE ALIAS/LANGUAGE=FORTRAN lse "The VAX Language-Sensitive Editor"
```

Causes the quoted string to appear when you type *lse* and then issue the EXPAND command when you are in the FORTRAN language environment.

See the section about defining an alias in the *VAX Language-Sensitive Editor and VAX Source Code Analyzer User Manual* for additional examples.

DEFINE COMMAND

Defines a user command or an abbreviation for an LSE command.

Format

DEFINE COMMAND *command-name* *value-string*

Parameters

command-name
Specifies the name to be defined as a command. A command name may contain up to 255 characters, but must begin with a letter, an underscore, or a dollar sign. After the first character, you may use any combination of alphanumeric characters, underscores, or dollar signs.

value-string
Specifies a quoted string containing an LSE command or the leading portion of an LSE command.

Description

With the DEFINE COMMAND command, you can define your own commands, or specify an abbbreviation for an LSE command. Before the command executes, LSE substitutes the specified value string for the command name.

To define a command for a sequence of commands, use the DO command inside the value string.

Related Commands

CALL
DO

Example

```
LSE> DEFINE COMMAND CLS "DO/TPU ""ERASE(CURRENT_BUFFER)"""
```

Associates the command name CLS with the command DO/TPU "ERASE(CURRENT_BUFFER)." After issuing this command, whenever you type CLS at the command prompt, LSE uses VAXTPU to clear all text from the current buffer.

DEFINE KEY

Binds an LSE command to a key.

Format

DEFINE KEY *key-specifier string*

Qualifiers	**Defaults**
/DIALOG	
/[NO]IF_STATE=GOLD	/NOIF_STATE
/LEARN	
/LEGEND=string	See text
/REMARK=(string, . . .)	
/STATE=GOLD	
/TOPIC_STRING=string	/TOPIC_STRING=No_topic

Qualifiers

/DIALOG
Specifies that a dialog box should be used to prompt the user for parameters and qualifier values. The command parameters are optional if this qualifier is specified. If command parameters and qualifiers are specified with the /DIALOG qualifier, the parameters and qualifiers are used to set the initial state of the dialog box.

/IF_STATE=GOLD
/NOIF_STATE (D)
Specifies that the key definition applies only to the GOLD (PF1) state.

/LEARN
Indicates that a sequence of keystrokes, called a learn sequence, defines the command to be bound to a key. You must type the keystroke sequence immediately after the command and end the sequence by specifying the END DEFINE command. If you are using the EVE keypad, CTRL/R is bound to the END DEFINE command by default. However, you do not

have to define a key to be the END DEFINE command to use the DEFINE KEY/LEARN command. When LSE records the learn sequence, the key being defined by the DEFINE KEY/LEARN command binds to the END DEFINE command. Therefore, you can press the key that you are defining to end the learn sequence.

When executing the stored sequence, LSE includes your responses to all prompts but does not prompt you again for such information as the string for a SEARCH command.

You may not use a learn sequence to enter a key definition while another key is in the process of being defined by another learn sequence.

/LEGEND=string
/LEGEND=?
Specifies the text that appears in the keypad diagram for this key. The string is centered in the figure for the key, or truncated if the string is too long for the figure.

If you do not specify the /LEGEND qualifier with a string, the default is /LEGEND=?.

/REMARK=(string, . . .)
Specifies the explanatory text that is displayed when you issue a SHOW KEY/FULL command.

/STATE=GOLD
Moves the functionality of the GOLD (PF1) key to the named key. You cannot specify the string parameter with the /STATE=GOLD qualifier.

/TOPIC_STRING=string
/TOPIC_STRING=No_Topic (D)
Specifies the string that the editor uses to retrieve help text for this key for display through the HELP /KEYPAD command.

If you do not specify a string with the /TOPIC_STRING qualifier, the default is /TOPIC_STRING=No_Topic.

Parameter

key-specifier

Specifies a keyword that indicates the key to be defined. If you use the DEFINE KEY command to change the definition of a key that was previously defined, LSE does not save the previous definition.

Table CD–1 lists the Language-Sensitive Editor's keynames and their VT200-type and VT100-type counterparts for the editing and auxiliary keypad. Table CD–2 lists the Language-Sensitive Editor's keynames and their VT200-type and VT100-type counterparts for the main keyboard keys.

As an alternative to using the /IF_STATE=GOLD qualifier, the key-specifier parameter accepts keynames prefixed with GOLD/. Also, you can specify control keys as CTRL/*x*, where *x* is an alphabetic character (A through Z).

string

Specifies an LSE command to be executed when the key is pressed. This is a required parameter unless you use the /LEARN qualifier; you cannot use the string parameter with either the /LEARN qualifier or the /STATE=GOLD qualifier.

Table CD–1: LSE Keynames for the Editing and Auxiliary Keypad

Keyname	VT200-type	VT100-type
PF1	PF1	PF1
PF2	PF2	PF2
PF3	PF3	PF3
PF4	PF4	PF4
KP0,KP1, . . . ,KP9	KP0,KP1, . . . ,KP9	KP0, KP1, . . . ,KP9
PERIOD	Keypad period (.)	Keypad period (.)
COMMA	Keypad comma (,)	Keypad comma (,)
MINUS	Keypad minus (–)	Keypad minus (–)
ENTER	Enter	Enter

(continued on next page)

Table CD–1 (Cont.): LSE Keynames for the Editing and Auxiliary Keypad

Keyname	VT200-type	VT100-type
UP	↑	↑
DOWN	↓	↓
LEFT	←	←
RIGHT	→	→
E1	Find/E1	
E2	Insert Here/E2	
E3	Remove/E3	
E4	Select/E4	
E5	Prev Screen/E5	
E6	Next Screen/E6	
HELP	Help/F15	
DO	Do/F16	
F7,F8, . . . ,F20	F7,F8 . . . ,F20	

Table CD–2: LSE Keynames for Keys on the Main Keyboard

Keyname	VT200-type	VT100-type
TAB_KEY	Tab	Tab
RET_KEY	Return	Return
DEL_KEY	⌫	Delete
LF_KEY	LF/F13	Line feed
BS_KEY	BS/F12	Backspace
SPACE_KEY	Space bar	Space bar
CTRL_A_KEY	CTRL/A	CTRL/A
CTRL_B_KEY	CTRL/B	CTRL/B

(continued on next page)

Table CD–2 (Cont.): LSE Keynames for Keys on the Main Keyboard

Keyname	VT200-type	VT100-type
.	.	.
.	.	.
.	.	.
CTRL_Z_KEY	CTRL/Z	CTRL/Z
NULL_KEY	CTRL/Space bar	CTRL/Space bar
FS_KEY	CTRL/\	CTRL/\
GS_KEY	CTRL/]	CTRL/]
RS_KEY	CTRL/~	CTRL/~
US_KEY	CTRL//	CTRL//

If you want to define a key to be lowercase, you must put the key specifier in lowercase and in quotes. However, GOLD and CTRL sequences are not case-sensitive. For example, CTRL/A and CTRL/a produce the same results. Also, GOLD/A is the same as GOLD/a.

The following combinations of the CTRL key and keyboard keys can be defined, but unless your terminal has the PASSALL characteristic set, you cannot execute your definitions of these keys:

CTRL/C	CTRL/O	CTRL/Q	CTRL/S
CTRL/T	CTRL/X	CTRL/Y	

If the following combinations of the CTRL key and keyboard keys are redefined, the new definition also affects the keyboard key corresponding to that combination. For example, if CTRL/I is redefined, then the TAB key also assumes that new definition.

CTRL/I	Tab
CTRL/M	Carriage return
CTRL/J	Line feed
CTRL/H	Backspace

If the first key pressed in response to the (Key:) prompt is a key that does not correspond to a printing key character, then LSE echoes the corresponding keyname. Tables CD–1 and CD–2 list keys that do not correspond to a printable character.

If the first key pressed is the GOLD key, then LSE waits for you to press a second key. LSE then echoes the key specifier for the key sequence. For example, if you press the GOLD key and then press the P key, LSE echoes GOLD/P.

Only the first key you press in response to the prompt (or the first two keys if the GOLD key is first) is handled in this special way. Subsequent input to the prompt is treated as though you typed in the text that LSE echos. CTRL/C, CTRL/Z, and the Return key required to end the input line are all handled in this way. Erasing all the text at the prompt (using CTRL/U or the DELETE command) causes LSE to interpret the next key input as the first key.

Description

The DEFINE KEY command associates an LSE command with a key. You may bind commands to control keys, to numeric keypad keys, and to the arrow keys on all keyboards. You may also bind a command to the sequence of the GOLD key followed by any keyboard key, where the GOLD key is the key defined to set the GOLD state (usually PF1). (On the VT200-series keyboard, you may also bind to the function (F) keys and the keys on the editing keypad.)

The HELP/KEYPAD command uses the values of the /LEGEND and /TOPIC qualifiers to build a keypad diagram for the keypad keys and to access help text for the keys. The SHOW KEY/FULL command displays the strings associated with the /LEGEND, /TOPIC_STRING, and /REMARK qualifiers.

The effect of a key can vary with its context. The DEFINE KEY command provides only for definitions for keys that are used in the work region.

DECwindows Interface Equivalent

DEFINE KEY/DIALOG
Pull-down menu: Customize --> Define key . . .

Related Commands

END DEFINE

Example

```
LSE> DEFINE KEY "GOLD/KP5" "GOTO TOP"
```

If the PF1 key sets the GOLD state, then the key sequence PF1-KP5 always issues a GOTO TOP command after you assign this definition.

See the section about defining keys in the *VAX Language-Sensitive Editor and VAX Source Code Analyzer User Manual* for additional examples.

DEFINE KEYWORDS

Defines the indicated keyword list.

Format

DEFINE KEYWORDS *keyword-list-name*

keyword [/DESCRIPTION=text]

.

.

.

keyword [/DESCRIPTION=text]

END DEFINE

Qualifier

/DESCRIPTION=text

Qualifier

/DESCRIPTION=text

Indicates the text to be associated with the individual keyword.

Parameters

keyword-list-name

Identifies the keyword list. The name must follow the rules applied to token names in LSE. You can then use the name as the value you specify for the /KEYWORDS qualifier to the DEFINE TAG command, as well as the parameter for the DELETE KEYWORDS, EXTRACT KEYWORDS, and SHOW KEYWORDS commands.

keyword
Names an individual keyword. Each keyword on the list must appear on a line by itself. You cannot use continuation characters between the lines for each keyword, but you can use a continuation character between a particular keyword and its associated qualifier.

Related Commands

DEFINE TAG
DELETE KEYWORDS
EXTRACT KEYWORDS
SHOW KEYWORDS

Example

```
DEFINE KEYWORDS author_name
   "Pat Jones"   /DESCRIPTION="Project Leader"
   "Chris Brown"
   "Leslie Green"
END DEFINE
```

Creates a keyword list named *author_names* and lists the individual names.

DEFINE LANGUAGE

Specifies the characteristics of a language.

Format

DEFINE LANGUAGE *language-name*

Qualifiers	**Defaults**
/CAPABILITIES=[NO]DIAGNOSTICS	/CAPABILITIES=NODIAGNOSTICS
/COMMENT=(specifier, . . .)	
/COMPILE_COMMAND=string	
/EXPAND_CASE=AS_IS	/EXPAND_CASE=AS_IS
/EXPAND_CASE=LOWER	/EXPAND_CASE=AS_IS
/EXPAND_CASE=UPPER	/EXPAND_CASE=AS_IS
/FILE_TYPES=(file-type[, . . .])	
/FORTRAN=[NO]ANSI_FORMAT	/FORTRAN=NOANSI_FORMAT
/[NO]HELP_LIBRARY=file-spec	/NOHELP_LIBRARY
/IDENTIFIER_CHARACTERS=string	
/INITIAL_STRING=string	
/LEFT_MARGIN=*n*	/LEFT_MARGIN=1
/OVERVIEW_OPTIONS=(MINIMUM_LINES=*m*, TAB_RANGE=(t1,t2))	
/PLACEHOLDER_DELIMITERS= (delimiter-specification[, . . .])	See text
/PUNCTUATION_CHARACTERS=string	/PUNCTUATION_CHARACTERS=",;()"
/[NO]QUOTED_ITEM=(QUOTES=string [,ESCAPES=string])	/NOQUOTED_ITEM
/RIGHT_MARGIN=*n*	/RIGHT_MARGIN=80
/TAB_INCREMENT=*n*	/TAB_INCREMENT=4
/TOPIC_STRING=string	
/VERSION=string	
/[NO]WRAP	/NOWRAP

Qualifiers

/CAPABILITIES=DIAGNOSTICS
/CAPABILITIES=NODIAGNOSTICS (D)
Specifies whether the compiler can generate diagnostic files.

/COMMENT=(specifier, . . .)
Specifies the character sequences of comments in the language. The specifiers are as follows:

- ASSOCIATED_IDENTIFIER=keyword

  Indicates the preferred association of comments to identifier. You can specify one of the following values:

  - NEXT—Indicates that comments should be associated with the next identifier.
  - PREVIOUS—Indicates that comments should be associated with the preceding identifier.

- BEGIN=list of quoted strings

  END=list of quoted strings

  Defines the character sequences that start and end bracketed comments. A bracketed comment begins and ends with explicit comment delimiters. (Note that the beginning and ending comment delimiters can be the same, but need not be.) The list provided with the specifiers BEGIN and END can be any of the following:

  - A string that is the one open comment sequence for the language. You must enclose this in quotes.
  - A parenthesized list of strings, each one of which can be an open comment sequence for the language. You must enclose each one in quotes.

  The list accompanying the BEGIN specifier must be consistent with the list accompanying the END specifier. If the BEGIN specifier lists a string, then the END specifier must also list a string.

  Bracketed comments are recognized by the formatting commands (see the ALIGN and FILL commands) and placeholder operations (see the ERASE PLACEHOLDER command and the /DUPLICATION qualifier of the DEFINE PLACEHOLDER command).

- TRAILING=list of quoted strings

  Defines the character sequence that introduces line-oriented comments. A line-oriented comment begins with a special character sequence (consisting of one or more characters) and ends at the end of the line. The list provided with the TRAILING specifier can be any of the following:

  - A string that is the one-line comment sequence for the language.
  - A list of strings enclosed in parentheses; each string can be a line-comment sequence for the language.

  Line comments are recognized by the formatting commands and placeholder operations, just as bracketed comments are.

- LINE=list of quoted strings

  Requires that the comment delimiter be the first character that is not blank on the line. The LINE specifier is particularly useful with block comments, such as the following:

  ```
  /*
  ** Here is the inside of a comment
  ** which has LINE="**" specified
  */
  ```

- FIXED=quoted string, column number

  Used for languages that require that a specific comment delimiter be placed in a specific column, such as FIXED=("*",1) for COBOL.

Note that for the specifier you cannot use any character that you used in the /PLACEHOLDER *delimiter-specification*.

/COMPILE_COMMAND=string
Specifies the default command string for the COMPILE command. (See the explanation of the *command-string* parameter in the COMPILE command entry.)

/EXPAND_CASE=AS_IS (D)
/EXPAND_CASE=LOWER
/EXPAND_CASE=UPPER
Specifies the case of the text of the inserted template. AS_IS specifies that the inserted template be expanded according to the case in the token or placeholder definition. LOWER and UPPER specify that the inserted template be expanded lowercase or uppercase, respectively.

/FILE_TYPES=(file-type[, . . .])
Specifies a list of file types that are valid for the language being defined. The file types must be enclosed in quoted strings. When LSE reads a file into a buffer, it sets the language for that buffer automatically if it recognizes the file type. For example, a FORTRAN file type (.FOR) sets the language to FORTRAN. Note that the period character must be included with the file type.

/FORTRAN=ANSI_FORMAT
/FORTRAN=NOANSI_FORMAT (D)
Specifies special processing for ANSI FORTRAN. Note that some commands behave differently when you use the /FORTRAN qualifier. Specifying NOANSI_FORMAT causes LSE to insert templates in non-ANSI (tab) format.

/HELP_LIBRARY=file-spec
/NOHELP_LIBRARY (D)
Specifies the HELP library where you can find help text for placeholders and tokens defined in this language. LSE applies the default file specification SYS$HELP:HELPLIB.HLB. If you want to access some HELP library other than SYS$HELP, you must supply an explicit device name.

/IDENTIFIER_CHARACTERS=string
Specifies the characters that may appear in token and alias names in that language. This list of characters is used in various contexts for the /INDICATED qualifier.

The list of identifier characters also determines what LSE considers to be a word. A word is a sequence of identifier characters, possibly followed by one or more blanks. All nonblank, nonidentifier characters are considered to be distinct words.

If you do not specify the /IDENTIFIER_CHARACTERS qualifier, LSE supplies the following values by default:

```
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ%$_0123456789"
```

/INITIAL_STRING=string
Specifies the initial text that is to appear in a newly created buffer.

/LEFT_MARGIN=n
/LEFT_MARGIN=1 (D)
/LEFT_MARGIN=CONTEXT_DEPENDENT
Specifies the left margin setting that is to be associated with the language.

If you specify CONTEXT_DEPENDENT as the column number, then LSE uses the indentation of the current line to determine the left margin when you use the /WRAP qualifier. When you use the FILL command, LSE uses the indentation of the first line of each selected paragraph to determine the left margin.

/OVERVIEW_OPTIONS=(MINIMUM_LINES=m, TAB_RANGE=(t1,t2))
Specifies both the minimum number of lines an overview line must hide and the range of acceptable tab increments.

The specifiers are as follows:

- MINIMUM_LINES=*m*

  Specifies the minimum number of lines an overview line must hide. The default is 1. For example, if the value of the parameter on MINIMUM_LINES is 5, then a line hides other lines only if there are at least five lines to hide. This specifier helps the user to avoid having very small source-line groups, and thus to avoid many expansion levels.

- TAB_RANGE=(t1,t2)

  The TAB_RANGE specifier indicates the range of tab values for which the adjustment definitions are valid. The default is (4,8). The second value must be at least twice the first value; both values must be positive. For example, if the tab range is (4,8), then LSE assumes that the adjustment definitions will work for any DEFINE LANGUAGE/TAB_INCREMENT value from 4 to 8 inclusive. If you specify a /TAB_INCREMENT value outside the tab range, then LSE recomputes indentation to make the adjustments work.

  For best performance, it is recommended that you avoid recomputation by choosing a range that covers reasonable values. The numbers specified for the DEFINE ADJUSTMENT/CURRENT and DEFINE ADJUSTMENT/SUBSEQUENT commands must work for any tab increment value in the tab range.

/PLACEHOLDER_DELIMITERS=(delimiter-specification[, . . .])
Specifies starting and ending strings that delimit placeholders. Placeholders can specify single constructs or lists of constructs. The delimiters for each type of placeholder are specified as a pair of quoted strings separated by commas and enclosed in parentheses.

The format of a delimiter specification is as follows:

keyword=(starting-string,ending-string)

Possible keywords are REQUIRED, REQUIRED_LIST, OPTIONAL, OPTIONAL_LIST, or PSEUDOCODE. If you do not use the PSEUDOCODE keyword, the default is NOPSEUDOCODE. The maximum length of these strings is seven characters.

The following is an example of a complete set of placeholder delimiter specifications:

```
/PLACEHOLDER_DELIMITERS = ( -
 REQUIRED =("{<",">}"), -
 REQUIRED_LIST=("{<",">}..."), -
 OPTIONAL =("[<",">]"), -
 OPTIONAL_LIST=("[<",">]..."), -
 PSEUDOCODE=("«" , "»"))
```

If any of the five keywords are not specified with the /PLACEHOLDER_DELIMITERS qualifier, LSE applies the following defaults:

```
/PLACEHOLDER_DELIMITERS = ( -
 REQUIRED =("{","}"), -
 REQUIRED_LIST=("{","}..."), -
 OPTIONAL =("[","]"), -
 OPTIONAL_LIST=("[","]..."), -
 NOPSEUDOCODE)
```

The placeholder delimiters that are accepted by each compiler are as follows:

ADA: { } , { }... , [ ] , [ ]... , « » , <| |>

BASIC: { } , { }... , [ ] , [ ]... , « » , << >>

BLISS: {~ ~} , {~ ~}... , [~ ~] , [~ ~]... , «» , <~ ~>

C: {@ @} , {@ @}... , [@ @] , [@ @]... , «» , <@ @>

COBOL: { } , { }... , [ ] , [ ]... , « » , << >>

FORTRAN: { } , { }... , [ ] , [ ]... , « » , << >>

PASCAL: *%{ }% , %{ }%... , %[ ]% , %[ ]%... , « » , %< >%*

PL1: { } , { }... , [ ] , [ ]... , « » , << >>

Note that for the specifier you cannot use any character that you used in the /COMMENT *specifier.*

/PUNCTUATION_CHARACTERS=string
/PUNCTUATION_CHARACTERS=",;()" (D)
Specifies the characters that are considered punctuation marks, or delimiters, in the language. When a placeholder name and its enclosing brackets are deleted, preceding white space is also deleted if there are punctuation characters to delimit the program constructs.

/QUOTED_ITEM=(QUOTES=string [,ESCAPES=string])
/NOQUOTED_ITEM (D)
Describes the syntax of certain language elements, such as strings, that require special handling for proper text formatting. LSE uses the /QUOTED_ITEM qualifier to detect comments properly. LSE does not acknowledge comment strings that occur within quoted items, nor does it acknowledge quoted elements that occur within comments.

The value of the /QUOTED_ITEM qualifier indicates the syntax of a quoted item. This value must be a keyword list. The keywords are as follows:

- QUOTES

  This keyword is required and must have an explicit value. The value must be a quoted string denoting all of the quote characters in the language. LSE assumes that quoted items begin and end with the same character.

- ESCAPES

  This keyword is optional. If given, then the value is required and must be a quoted string containing the escape characters for quoted items. Some languages use escape characters to insert quote characters into strings. For example, C uses the backslash (\ ) as an escape character. If you omit this keyword, then LSE assumes that the language inserts quote characters into strings by doubling them.

/RIGHT_MARGIN=n
/RIGHT_MARGIN=80 (D)
Specifies the right margin setting that is to be associated with the language. By default, the right margin is set at column 80.

/TAB_INCREMENT=n
/TAB_INCREMENT=4 (D)
Specifies that tab stops be set every *n* columns, beginning with column 1.

/TOPIC_STRING=string
Specifies a prefix string to be concatenated to the /TOPIC_STRING qualifier specified in a placeholder or token definition before LSE looks up the help text for that placeholder or token. (Typically, this is the name of the language in the HELP library.)

/VERSION=string
Specifies a string that represents the version number of the tokens and placeholders associated with this language. You use the SHOW LANGUAGE command to display this string.

/WRAP
/NOWRAP (D)
Specifies whether the ENTER SPACE command (bound to the space bar by default) should wrap text when there is too much to fit on the current line. The /NOWRAP qualifier disables such text wrapping.

Parameter

language-name
Specifies the name of the language whose characteristics are to be defined.

Description

The DEFINE LANGUAGE command specifies a language so that LSE can properly recognize language-specific text characteristics.

After you have specified these language characteristics by using the DEFINE LANGUAGE command, you can use the MODIFY LANGUAGE command when you want to make subsequent changes.

Related Commands

DELETE LANGUAGE
EXTRACT LANGUAGE
MODIFY LANGUAGE
SET LANGUAGE
SHOW LANGUAGE

Examples

```
1.  DEFINE LANGUAGE ADA -
      /CAPABILITIES=DIAGNOSTICS -
      /COMPILE_COMMAND="ADA" -
      /FILE_TYPES=(.ADA) -
      /IDENTIFIER_CHARACTERS= -
          "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ$_0123456789" -
      /INITIAL_STRING="{compilation_unit}" -
      /COMMENT=(TRAILING="--") -
      /PLACEHOLDER_DELIMITERS=( -
        REQUIRED=("{","}"), -
        REQUIRED_LIST=("{","}..."), -
        OPTIONAL=("[","]"), -
        OPTIONAL_LIST=("[","]...") ) -
      /PUNCTUATION_CHARACTERS=",;()*.'" -
      /QUOTED_ITEM=(QUOTES="""'") -
      /TAB_INCREMENT=4 -
      /TOPIC_STRING="ADA Language_Topics"
```

Defines characteristics of the Ada language.

```
2.  DEFINE LANGUAGE PASCAL -
      /CAPABILITIES=DIAGNOSTICS -
      /COMMENT=(BEGIN=("{","(*"),END=("}","*)")) -
      /COMPILE_COMMAND="PASCAL " -
      /FILE_TYPES=(.PAS) -
      /IDENTIFIER_CHARACTERS= -
          "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ$%_0123456789" -
      /INITIAL_STRING="%{compilation_unit}%" -
      /PLACEHOLDER_DELIMITERS=( -
        REQUIRED=("%{","}%"), -
        REQUIRED_LIST=("%{","}%..."), -
        OPTIONAL=("%[","]%"), -
        OPTIONAL_LIST=("%[","]%...") ) -
      /PUNCTUATION_CHARACTERS=",;:()[]{}.'" -
      /QUOTED_ITEM=(QUOTES="""'") -
      /TAB_INCREMENT=4 -
      /TOPIC_STRING="PASCAL "
```

Defines characteristics of the Pascal language.

See the sections about language definition in the *VAX Language-Sensitive Editor and VAX Source Code Analyzer User Manual* for additional examples.

DEFINE PACKAGE

Defines a subroutine package for which subroutine call templates are automatically generated.

Format

DEFINE PACKAGE *package-name*

Qualifiers
/HELP_LIBRARY=file-spec
/LANGUAGE=(language [, . . .])
/PARAMETER_EXPAND=TPU_procedure_prefix
/ROUTINE_EXPAND=TPU_procedure_prefix
/TOPIC_STRING=string

Qualifiers

/HELP_LIBRARY=file-spec
Specifies the HELP file (.HLB file) where you can find help text for placeholders and tokens defined for this package. If omitted, then no HELP file is associated with the package.

LSE applies the default file specification SYS$HELP:HELPLIB.HLB. If you want to access some device or directory other than SYS$HELP, you must supply an explicit device and directory name.

/LANGUAGE=(language1 [, . . .])
Specifies the languages from which LSE can use package entries. If you do not specify a language, then LSE uses the language of the current buffer. If no language is associated with the current buffer, then an error occurs.

/PARAMETER_EXPAND=TPU_procedure_prefix
Lets you customize calling sequences. Normally LSE uses a fixed algorithm to produce the appropriate placeholder definitions from the DEFINE PARAMETER command. If the default algorithm is inadequate, then

you can supply explicit VAXTPU procedures to produce the DEFINE PLACEHOLDER command that you want.

The argument provided with the /PARAMETER_EXPAND qualifier must be the prefix of a VAXTPU procedure name. Whenever LSE expands a parameter, it concatenates this prefix and the current language name, and looks for a VAXTPU procedure by that name. For details, see Appendix D.

/ROUTINE_EXPAND=TPU_procedure_prefix

Lets you customize calling sequences. Normally LSE uses a fixed algorithm to produce the appropriate token and placeholder definitions from the DEFINE ROUTINE command. If the default algorithm is inadequate, then you can supply explicit VAXTPU procedures to produce the DEFINE PLACEHOLDER or DEFINE TOKEN commands that you want.

The argument provided with the /ROUTINE_EXPAND qualifier must be the prefix of a VAXTPU procedure name. Whenever LSE expands a routine, it concatenates this prefix and the current language name, and looks for a VAXTPU procedure by that name. For example, if you specify /ROUTINE_EXPAND=my_routine_expand_ and the current language is FORTRAN, then LSE looks for a VAXTPU procedure named *my_routine_expand_fortran*. For details, see Appendix D.

/TOPIC_STRING=string
Specifies a prefix string to be concatenated to the TOPIC_STRING specified for tokens and placeholders associated with the routine or parameter definitions. If omitted, then the null string is used as the topic string. LSE uses the topic string to look up help text for the package.

Parameter

package-name
Specifies the name of the package being defined.

DEFINE PACKAGE

Description

The DEFINE PACKAGE command defines a subroutine package for which subroutine-call templates are automatically generated. Packages can contain routine definitions, which describe calls to subroutines, and parameter definitions, which describe parameters for subroutine calls.

Related Commands

DEFINE PARAMETER
DEFINE ROUTINE
DELETE PACKAGE
EXTRACT PACKAGE
SHOW PACKAGE

Example

```
DEFINE PACKAGE system_services -
    /LANGUAGES =(BASIC,C,COBOL,FORTRAN,PLI) -
    /HELP_LIBRARY = HELPLIB -
    /TOPIC_STRING = "system_services" -
    /ROUTINE_EXPAND = "LSE$PKG_EXPAND_ROUT_" -  ! Special routines for
    /PARAMETER_EXPAND = "LSE$PKG_EXPAND_PARM_"  ! system services

DEFINE ROUTINE sys$add_holder -
    /PACKAGE = system_services -
    /DESCRIPTION = "Add Holder Record To The Rights Database" -
     id/BY_VALUE, -
     holder/BY_REFERENCE, -
     attrib/BY_VALUE/OPTIONAL -
DEFINE PARAMETER id -
    /PACKAGE = system_services -
DEFINE PARAMETER holder -
    /PACKAGE = system_services -
DEFINE PARAMETER attrib -
    /PACKAGE = system_services -
```

Shows the incorporation of the DEFINE PACKAGE command in a complete package definition, along with DEFINE PARAMETER and DEFINE ROUTINE commands.

DEFINE PARAMETER

Defines a parameter within a package.

Format

DEFINE PARAMETER *param-name*

Qualifier
/PACKAGE=package-name

Qualifier

/PACKAGE=package-name
Specifies the name of the package with which the parameter is associated.

Parameter

param-name
Specifies the name of the parameter. This name must be unique among the tokens of any language from which the package is used.

Description

The DEFINE PARAMETER command defines a parameter within a package. That parameter can be associated with more than one routine by means of the DEFINE ROUTINE command.

Related Commands

DEFINE PACKAGE
DEFINE ROUTINE
DELETE PARAMETER
EXPAND
EXTRACT PARAMETER
SHOW PARAMETER

Example

```
DEFINE PACKAGE system_services -
    /LANGUAGES =(BASIC,C,COBOL,FORTRAN,PLI) -
    /HELP_LIBRARY = HELPLIB -
    /TOPIC_STRING = "system_services" -
    /ROUTINE_EXPAND = "LSE$PKG_EXPAND_ROUT_" -  ! Special routines for
    /PARAMETER_EXPAND = "LSE$PKG_EXPAND_PARM_"  ! system services

DEFINE ROUTINE sys$add_holder -
    /PACKAGE = system_services -
    /DESCRIPTION = "Add Holder Record To The Rights Database" -
     id/BY_VALUE, -
     holder/BY_REFERENCE, -
     attrib/BY_VALUE/OPTIONAL -
DEFINE PARAMETER id -
    /PACKAGE = system_services -
DEFINE PARAMETER holder -
    /PACKAGE = system_services -
DEFINE PARAMETER attrib -
    /PACKAGE = system_services -
```

Shows the incorporation of the DEFINE PARAMETER command into a complete package definition, along with DEFINE PACKAGE and DEFINE ROUTINE commands.

DEFINE PLACEHOLDER

Creates a placeholder for use with a specific language and establishes the characteristics of that placeholder.

Format

DEFINE PLACEHOLDER *placeholder-name*

placeholder body

END DEFINE

or

DEFINE PLACEHOLDER *placeholder-name*

/PLACEHOLDER= *other-placeholder*

Qualifiers	**Defaults**
/[NO]AUTO_SUBSTITUTE	/NOAUTO_SUBSTITUTE
/DESCRIPTION=string	
/DUPLICATION=specifier	/DUPLICATION=CONTEXT_DEPENDENT
/LANGUAGE=language-name	
/LEADING=string	
/PLACEHOLDER=other-placeholder	
/[NO]PSEUDOCODE	/PSEUDOCODE
/SEPARATOR=string	
/TOPIC_STRING=string	
/TRAILING=string	
/TYPE=type-specifier	/TYPE=NONTERMINAL

Qualifiers

/AUTO_SUBSTITUTE
/NOAUTO_SUBSTITUTE (D)
Specifies whether you want the next placeholder with this name to be replaced with the same text you typed over the current placeholder.

/DESCRIPTION=string
Specifies a single line of text to be displayed along with the placeholder name when the placeholder name appears in a menu during an EXPAND operation.

/DUPLICATION=specifier
/DUPLICATION=CONTEXT_DEPENDENT (D)
Specifies the type of duplication to be performed when the placeholder is duplicated (either by expanding it or by typing over it). The specifier is one of the following keywords:

- CONTEXT_DEPENDENT

  If the placeholder is the only item within its segment (that is, if it is either the only item before or the only item within a trailing comment), then LSE duplicates it vertically (see the VERTICAL keyword in this list). Otherwise, LSE duplicates it horizontally. White space may precede or follow the placeholder.
- HORIZONTAL

  LSE places the duplicate immediately to the right of the original. If you specify a separation string, LSE places the string between the original and the duplicate.
- VERTICAL

  LSE places the duplicate on the next line, immediately under the original. If a separation string is specified, LSE places it at the end of the original. If the original placeholder is in the commented segment of the line, then LSE also duplicates the comment delimiters directly underneath the delimiters in the original line. If necessary, LSE adds close comment delimiters to the original line to close a bracketed comment on that line.

/LANGUAGE=language-name
Specifies the language associated with the placeholder. By default, the new placeholder is defined for use with the current buffer's language.

/LEADING=string
Specifies any leading text to be associated with the placeholder. The ERASE PLACEHOLDER command recognizes this text and erases it along with the placeholder. The leading text must not have any trailing blank spaces because the ERASE PLACEHOLDER command always skips over such spaces.

/PLACEHOLDER=other-placeholder
Specifies the name of another defined placeholder from which this placeholder inherits its definition. A placeholder defined with the /PLACEHOLDER qualifier may not be named on the /PLACEHOLDER qualifier of any other definition. The /PLACEHOLDER qualifier is mutually exclusive with all other qualifiers except the /LANGUAGE qualifier.

/PSEUDOCODE (D)
/NOPSEUDOCODE
Specifies whether pseudocode can be entered at a specific placeholder. If you specify the /NOPSEUDOCODE qualifier for a placeholder, that placeholder cannot be used with pseudocode.

/SEPARATOR=string
Specifies the string that separates each duplication of the placeholder. See the /DUPLICATION qualifier description.

/TOPIC_STRING=string
Specifies a quoted string that LSE uses to retrieve help text for this placeholder. This string is appended to the string you specify with the /TOPIC_STRING qualifier of the DEFINE LANGUAGE command to form the complete string of topics that LSE uses for looking up the help text for this placeholder.

/TRAILING=string

Specifies any trailing text to be associated with the placeholder. The ERASE PLACEHOLDER command recognizes this text and erases it along with the placeholder. The trailing text must not have any leading blank spaces because the ERASE PLACEHOLDER command always skips over such spaces.

/TYPE=type-specifier
/TYPE=NONTERMINAL (D)

Specifies the kind of placeholder being defined. The type specifier may be NONTERMINAL, MENU, or TERMINAL.

Parameters

placeholder-name

Specifies the name of the placeholder being defined. A placeholder name must be unique within a language and can be a quoted string. To redefine an existing placeholder, you must first delete it using the DELETE PLACEHOLDER command.

placeholder body

Is the body of the placeholder being defined. The interpretation of the placeholder body depends on the type of placeholder. LSE displays the body of a terminal placeholder when you attempt to expand the placeholder. Note that displaying this text does not replace the terminal placeholder and its delimiters.

The body of a nonterminal placeholder is the text of the placeholder expansion; when a nonterminal placeholder is expanded, the placeholder name and enclosing delimiters are replaced with the text of the placeholder body.

A nonterminal placeholder can have more than one quoted string in each body line. For the expansion of the placeholder, you can set the indentation of each string by using the /INDENTATION qualifier and its associated keywords.

Each quoted string in the body line of a nonterminal placeholder can take the qualifier and keywords described in the following section.

Nonterminal Body Qualifier

/INDENTATION=(keyword1 [,integer1, keyword2])

keyword1

You can specify any of the following options for *keyword1*:

Option	Description
EXPAND	Indents the string to the column of the first character of the nonterminal placeholder being expanded. This is the default value if the first body line is not a null string.
CURRENT	Indents the string to the indentation of the line containing the placeholder or token. This is the default value if the first body line is a null string.
PREVIOUS	Indents the string to the indentation of the line before the line containing the placeholder or token.
FIXED	Indents the string to the specified column.

integer1

You can specify any integer for the *integer1* option. The default is 0. The integer is added to the column position as specified by *keyword1* and adjusts the indentation by that number of columns. The integer can be negative. When the value for *keyword1* is FIXED, *integer1* specifies the column position at which to put body text; it must be positive.

keyword2

You can specify any of the following options for *keyword2*:

Option	Description
TAB	Specifies that *integer1* should be interpreted as specifying an adjustment in terms of tab increments rather than columns. *Integer1* is multiplied by the tab increment for the buffer before it is added to the column specified by *keyword1*.
SPACE	Specifies that *integer1* should be interpreted as specifying an adjustment in terms of spaces. This is the default.

Note that you cannot specify *keyword2* when *keyword1* has a FIXED value.

If there is more than one quoted string in a body line, a comma must separate the strings. For FORTRAN, if the body line is inside of a comment or there is a tab in the body lines, the /INDENTATION qualifier and associated keywords do not take effect for the first quoted string for each body line.

For more information about the use of the /INDENTATION qualifier, see the examples for the EXPAND command.

Each line of the body of a menu placeholder represents one option in the menu. An option can be a string of text, a placeholder name, or a token name. If the option is a string of text, it must appear in quotes. If the option is a placeholder name or a token name and does not appear in quotes, that placeholder name or token name appears in uppercase letters in the menu display. For a placeholder name or token name to appear in lowercase letters in a menu, you must enter the placeholder name or token name as a lowercase quoted string.

Each line in the body of a menu placeholder may take one or more of the following qualifiers:

Menu Body Qualifiers	Default
/DESCRIPTION=string	
/[NO]LIST	/NOLIST
/PLACEHOLDER	
/TOKEN	

/DESCRIPTION=string

Specifies a description string that is displayed in the right-hand column of the menu. If this qualifier is omitted, then LSE gets the description string from the corresponding definition if the line has either the /TOKEN or the /PLACEHOLDER qualifier. If neither /TOKEN nor /PLACEHOLDER is specified, then the line is a literal string and the value of the /DESCRIPTION string defaults to the empty string.

/LIST
/NOLIST (D)

Specifies whether the delimiters for the placeholder should be list delimiters or not. Use this qualifier only in conjunction with the /PLACEHOLDER qualifier.

/PLACEHOLDER

Specifies that the name or string is the name of a placeholder in the language. This qualifier is mutually exclusive with the /TOKEN qualifier.

/TOKEN

Specifies that the name or string is the name of a token in the language. This qualifier is mutually exclusive with the /PLACEHOLDER and /[NO]LIST qualifiers.

Description

The DEFINE PLACEHOLDER command creates and establishes the characteristics of a placeholder for use with a particular language. A placeholder definition consists of a DEFINE PLACEHOLDER command followed by a placeholder body (which may occupy more than one line). If you do not specify the /PLACEHOLDER qualifier, you must end the placeholder body with an END DEFINE command.

Subsequently, you can use the new placeholder with the EXPAND and HELP/LANGUAGE commands.

Related Commands

DEFINE TOKEN
DELETE PLACEHOLDER
END DEFINE
EXPAND
EXTRACT PLACEHOLDER
HELP/INDICATED
SHOW PLACEHOLDER

Examples

1.
```
DEFINE PLACEHOLDER parameter -
     /LANGUAGE = EXAMPLE -
     /DESCRIPTION = "Parameter name"
     /DUPLICATION = HORIZONTAL -
     /SEPARATOR = ", " -
     /TYPE = TERMINAL
     "A string of letters and digits starting with a letter."
   END DEFINE
```

Creates a placeholder named *parameter* and establishes its characteristics.

2.
```
DEFINE PLACEHOLDER "#IF" -
     /LANGUAGE=C -
     /TYPE=NONTERMINAL -

     "#if {@constant expression@}"/INDENTATION=(FIXED,1)
     "[@#else_clause@]"/INDENT=(FIXED,1)
     "#endif"/INDENTATION=(FIXED,1)

END DEFINE
```

The /INDENTATION=(FIXED,1) qualifier puts the body text at column 1 while the expanding operation is performed. With the definitions in this example, the expanded placeholder [@#IF@] is as follows:

```
#if {@constant expression@}
[@#else_clause@]
#endif
```

For additional examples, see the sections about placeholder definitions and language elements in the *VAX Language-Sensitive Editor and VAX Source Code Analyzer User Manual.*

DEFINE ROUTINE

Defines templates for a routine contained within a subroutine package.

Format

DEFINE ROUTINE *routine-name [parameter, . . .]*

Qualifiers
/DESCRIPTION=string
/PACKAGE=package-name
/TOPIC_STRING=string

Qualifiers

/DESCRIPTION=string
Specifies a single line of text to be displayed along with the routine name when the routine name appears in a menu during an EXPAND operation. The string is also passed to the /ROUTINE_EXPAND procedure, if any. (The default algorithm for producing routine calls from DEFINE ROUTINE commands does not make use of this value.)

/PACKAGE=package_name
Specifies the name of the package with which the routine is associated. You must specify this qualifier.

/TOPIC_STRING=string
Specifies a quoted string that LSE uses to retrieve help text for this routine.

Parameters

routine-name
Specifies the name of the routine. Routine names must be unique within a package. Furthermore, routine names may not conflict with any token names used by LSE for any language using the package.

parameter, . . .
Specifies the names of the parameters of the routine. These parameters must be defined (using the DEFINE PARAMETER command) prior to expanding an instance of a call on this routine. However, the parameters do not need to be defined prior to the DEFINE ROUTINE command. If you omit this qualifier, then the routine is presumed to have no parameters.

The following qualifiers are position-sensitive; they may be used only with the list of parameters to the routine.

/BY_VALUE

Indicates that the parameter is passed by value.

/BY_REFERENCE

Indicates that the parameter is passed by address.

/BY_DESCRIPTOR

Indicates that the address of the parameter descriptor is passed.

/[NO]OPTIONAL

Specifies whether the parameter is required or optional. The default is /NOOPTIONAL.

The /BY_VALUE, /BY_REFERENCE, and /BY_DESCRIPTOR qualifiers are mutually exclusive. These qualifiers are used primarily for languages, such as COBOL, that require explicit specification of passing mechanisms for routine calls.

Description

The DEFINE ROUTINE command defines templates for a routine contained within a subroutine package. This command makes the routine known as an element of a package. The first time that the routine name is expanded, LSE generates an appropriate template and simulates a corresponding DEFINE TOKEN command. Thus, you can expand and unexpand routines in the same manner as tokens. Note, however, that commands such as SHOW TOKEN do not operate on tokens defined from routines; instead, you should use the appropriate routine commands, such as SHOW ROUTINE and EXTRACT ROUTINE.

Related Commands

DEFINE PACKAGE
DEFINE PARAMETER
DELETE ROUTINE
EXPAND
EXTRACT ROUTINE
SHOW ROUTINE

Example

```
DEFINE PACKAGE system_services -
       /LANGUAGES =(BASIC,C,COBOL,FORTRAN,PLI) -
       /HELP_LIBRARY = HELPLIB -
       /TOPIC_STRING = "system_services" -
       /ROUTINE_EXPAND = "LSE$PKG_EXPAND_ROUT_" -  ! Special routines for
       /PARAMETER_EXPAND = "LSE$PKG_EXPAND_PARM_"  ! system services

   DEFINE ROUTINE sys$add_holder -
       /PACKAGE = system_services -
       /DESCRIPTION = "Add Holder Record To The Rights Database" -
        id/BY_VALUE, -
        holder/BY_REFERENCE, -
        attrib/BY_VALUE/OPTIONAL
   DEFINE PARAMETER id -
       /PACKAGE = system_services
   DEFINE PARAMETER holder -
       /PACKAGE = system_services
   DEFINE PARAMETER attrib -
       /PACKAGE = system_services
```

Shows the incorporation of the DEFINE ROUTINE command into a complete package definition, along with DEFINE PACKAGE and DEFINE PARAMETER commands.

DEFINE TAG

Defines the specified tag.

Format

DEFINE TAG *tag-name*

Qualifiers	**Defaults**
/EMPTY=string-list	/EMPTY="None"
/KEYWORDS=keyword-list-name	
/LANGUAGE=language-name	
/SUBTAGS=tag-list	
/TYPE=type-keyword	/TYPE=TEXT

Qualifiers

/EMPTY=string-list
/EMPTY="None" (D)
Specifies one or more strings that indicate that a use of the structured tag has no subtags. If you do not specify the /EMPTY qualifier, then there will be no way to explicitly indicate that an occurrence of the tag is empty. You can always use implicitly empty tags by starting a new top-level tag after the current top-level tag, or by terminating the comment block.

You use this qualifier only with the /TYPE=STRUCTURED case.

/KEYWORDS=keyword-list-name
Defines the keywords that you can use with this tag. You must specify the *keyword-list-name* parameter by using the DEFINE KEYWORDS command. If you specify /KEYWORDS=*, this indicates that any keyword is allowed and no checking of keywords is to be done.

You use this qualifier only with the /TYPE=KEYWORD case.

/LANGUAGE=language-name
Specifies the language associated with the tag being defined. If you do not specify a language, the default is the language of the current buffer.

/SUBTAGS=tag-list
Indicates the subtags that can appear in a structured tag. The special case /SUBTAGS=* indicates that any tag is allowed. For example, you would use this special case for the PARAMETERS tag.

You use this qualifier only with the /TYPE=STRUCTURED case.

/TYPE=type-keyword
Indicates the type of the tag. You can specify any one of the following types:

Keyword Type	Description
TEXT	Ordinary text tag (default)
KEYWORD	List of keywords to be parsed at compile time
STRUCTURED	Sequence of zero or more subtags

Parameter

tag-name
Specifies the name of the tag being defined. The tag name must consist only of alphanumeric characters, the dollar sign ($), or the underscore (_), and may contain embedded blanks. Tag names are case-insensitive. If you include embedded blanks, place the name inside quotation marks.

Description

The DEFINE TAG command defines the indicated tag. Tags are headings embedded inside comments for use with design reports. You can save the definition in an environment file and direct the compiler to process tags with the /DESIGN qualifier.

For more information about how to use tags, see the section about design information in the *VAX Language-Sensitive Editor and VAX Source Code Analyzer User Manual.*

Related Commands

DELETE TAG
EXTRACT TAG
SHOW TAG

Examples

1.
```
LSE> DEFINE TAG "functional description"
```

Defines the tag *functional description* and indicates that the tag is an ordinary text tag.

2.
```
LSE> DEFINE TAG parameters /TYPE=STRUCTURED /SUBTAGS=*
_LSE> /EMPTY=("None", "Omitted")
```

Defines the tag *parameters*, specifies that the tag type is STRUCTURED, and indicates that any tag is allowed. The /EMPTY=("None", "Omitted") qualifier indicates that you can use either the word *None* or the word *Omitted* in your programs to explicitly indicate that the tag has no subtag values.

DEFINE TOKEN

Defines an editing token for use with the EXPAND command.

Format

DEFINE TOKEN *token-name*

token body

END DEFINE

or

DEFINE TOKEN *token-name*

/PLACEHOLDER= *placeholder-name*

Qualifiers
/DESCRIPTION=string
/LANGUAGE=language-name
/PLACEHOLDER=placeholder-name
/TOPIC_STRING=string

Qualifiers

/DESCRIPTION=string
Specifies some text to be displayed along with the token name when the token name appears in a menu during an EXPAND operation or in a SHOW TOKEN display.

/LANGUAGE=language-name
Specifies the language associated with the token. By default, the token is defined for use with the current language.

/PLACEHOLDER=placeholder-name

Specifies the name of a defined placeholder that expands in place of the token. The token gets its description, topic string, and body from the defining placeholder.

Note that the /PLACEHOLDER qualifier is mutually exclusive with the /DESCRIPTION and /TOPIC_STRING qualifiers and the END DEFINE command must not be used on the DEFINE TOKEN command when /PLACEHOLDER is specified. No token body is specified with the /PLACEHOLDER qualifier.

/TOPIC_STRING=string

Specifies a quoted string that LSE uses to retrieve help text for this token. This string is appended to the /TOPIC_STRING qualifier specified in the DEFINE LANGUAGE command to form the complete string of topics that LSE uses to look up the help text for this token.

Parameter

token-name

Specifies the name for the token being defined. Each token for a particular language must have a unique name. Token and alias names must not conflict. A token name can be any character including a blank space, but not a leading or trailing space.

token body

Is the text of the token expansion. When the token is expanded, the token name is replaced with the text of the token body. A token can have more than one quoted string in each body line. For the expansion of the token, you can set the indentation of each string by using the /INDENTATION qualifier and its associated keywords described in the following section:

Each quoted string in the body line of a token can take the qualifier and keywords described in the following section.

Nonterminal Body Qualifier

/INDENTATION=(keyword1 [,integer1, keyword2])

keyword1

You can specify any of the following options for *keyword1*:

Option	Description
EXPAND	Indents the string to the column of the first character of the nonterminal placeholder being expanded. This is the default value if the first body line is not a null string.
CURRENT	Indents the string to the indentation of the line containing the placeholder or token. This is the default value if the first body line is a null string.
PREVIOUS	Indents the string to the indentation of the line before the line containing the placeholder or token.
FIXED	Indents the string to the specified column.

integer1

You can specify any integer for the *integer1* option. The default is 0. The integer is added to the column position as specified by *keyword1* and adjusts the indentation by that number of columns. The integer can be negative. When the value for *keyword1* is FIXED, *integer1* specifies the column position at which to put body text and must be positive.

keyword2

You can specify any of the following options for *keyword2*:

Option	Description
TAB	Specifies that *integer1* should be interpreted as specifying an adjustment in terms of tab increments rather than columns. *Integer1* is multiplied by the tab increment for the buffer before it is added to the column specified by *keyword1*.
SPACE	Specifies that *integer1* should be interpreted as specifying an adjustment in terms of spaces. This is the default.

Note that you cannot specify *keyword2* when *keyword1* has a FIXED value.

If there is more than one quoted string in a body line, a comma must separate the strings. For FORTRAN, if the body line is inside of a comment or there is a tab in the body lines, the /INDENTATION qualifier and associated keywords do not take effect for the first quoted string for each body line.

For more information about the use of the /INDENTATION qualifier, see the examples for the EXPAND command.

Description

The DEFINE TOKEN command defines an editing token for use with the EXPAND command. When you issue the EXPAND command while the cursor is positioned immediately after the token name or an abbreviation of the token name, LSE replaces the input string with the body of the token.

Related Commands

DELETE TOKEN
EXPAND
EXTRACT TOKEN
SHOW TOKEN

Examples

1.
```
DEFINE TOKEN ASSIGNMENT -
     /LANGUAGE = EXAMPLE -
     /DESCRIPTION = "Assignment statement"
     "{identifier} = {expression}"
   END DEFINE
```

   Creates a token named *ASSIGNMENT* and establishes its characteristics.

2.
```
DEFINE TOKEN Parameter_template
   /PLACEHOLDER = Parameter
```

   Creates a token named *Parameter_template.* When you expand this token, LSE substitutes the placeholder named *Parameter* for the token.

3.
```
DEFINE TOKEN { -
   /LANGUAGE=C

   "{"/INDENTATION=EXPAND
   "[@block declaration@]..."/INDENTATION=(CURRENT,1,TAB)
   ""/INDENTATION=CURRENT
   "{@statement@}..."/INDENTATION=(CURRENT,1,TAB)
   "}"/INDENTATION=CURRENT

END DEFINE
```

The /INDENTATION=(CURRENT,1,TAB) qualifier indents the body text at the current indentation plus the number of spaces equivalent to one tab increment for the language. Specifying /INDENTATION=EXPAND indents the body text at the cursor's position. Specifying /INDENTATION=CURRENT replaces the body text at the current indentation level. With these definitions, if you expand the token { in the following example

```
if (a == b) {
```

it becomes as follows:

```
if (a == b) {
    [@block declaration@]...

    {@statement@}...
}
```

For additional examples, see the sections about token definitions and defining language elements in the *VAX Language-Sensitive Editor and VAX Source Code Analyzer User Manual.*

DELETE ADJUSTMENT

Removes a name from the list of adjustments associated with a language.

Format

DELETE ADJUSTMENT *adjustment-name*

Qualifier
/LANGUAGE=language-name

Qualifier

/LANGUAGE=language-name
Names the language associated with the adjustment being deleted. By default, LSE deletes the adjustment from the set of adjustments defined for the current language. By using the /LANGUAGE qualifier, you can delete adjustments from other languages as well.

Parameter

adjustment-name
Specifies the name of the adjustment to be deleted.

Description

The DELETE ADJUSTMENT command removes a specified name from the list of adjustments associated with a language.

Related Commands

DEFINE ADJUSTMENT
EXTRACT ADJUSTMENT
SHOW ADJUSTMENT

Example

```
LSE> DELETE ADJUSTMENT/LANGUAGE=EXAMPLE then
```

Removes the adjustment named *then* from the list of adjustments associated with the language EXAMPLE.

DELETE ALIAS

Deletes the definition of an alias name.

Format

DELETE ALIAS *alias-name*

Qualifier
/LANGUAGE=language-name

Qualifier

/LANGUAGE=language-name
Specifies the name of the language in which the alias is defined. The default is the current language.

Parameter

alias-name
Specifies the alias name to be deleted.

Description

The DELETE ALIAS command cancels the definition of an alias name established by a previous DEFINE ALIAS command.

Related Commands

DEFINE ALIAS

Example

```
LSE> DELETE ALIAS lse
```

Cancels the definition of the alias named *lse*.

DELETE BUFFER

Deletes a buffer.

Format

DELETE BUFFER *[buffer-name]*

Parameter

buffer-name
Indicates which buffer is to be deleted. The default is the current buffer.

Description

The DELETE BUFFER command deletes the specified buffer. If the buffer is being displayed, LSE replaces it with another buffer. You cannot delete system buffers.

If the indicated buffer has been modified and is not read-only, LSE prompts you to answer Y if you want to continue the DELETE BUFFER operation; otherwise answer N.

Related Commands

GOTO BUFFER
GOTO FILE
SHOW BUFFER

Example

```
LSE> DELETE BUFFER USER.BUF
```

Deletes the buffer named *USER.BUF*.

DELETE COMMAND

Deletes the definition of the indicated user-defined command.

Format

DELETE COMMAND *command-name*

Parameter

command-name
Specifies the command to be deleted.

Description

The DELETE COMMAND command cancels the definition of a command previously established by a DEFINE COMMAND command.

Related Commands

DEFINE COMMAND

Example

```
LSE> DELETE COMMAND XYZ
```

Cancels the definition of the user-defined command *XYZ*.

DELETE KEY

Deletes the specified key definition.

Format

DELETE KEY *key-specifier*

Qualifier	**Default**
/[NO]IF_STATE=GOLD	/NOIF_STATE

Qualifier

/IF_STATE=GOLD
/NOIF_STATE (D)
Specifies that the key definition to be deleted is for the GOLD state. The default is to delete the key definition for the default state.

Parameter

key-specifier
Indicates the keyword or single printing character for the key to be deleted. Valid key-specifiers include all keynames recognized by the DEFINE KEY command.

Description

The DELETE KEY command cancels a key definition established by a previous DEFINE KEY command. If the key is a printing key, LSE restores the original function of inserting a printing character at the current cursor position; otherwise, the key is undefined.

Related Commands

DEFINE KEY
SHOW KEY

Example

```
LSE>  DELETE KEY KP7
```

Deletes the definition for the KP7 key (the 7 key on the numeric keypad).

DELETE KEYWORDS

Cancels the indicated keywords list definition.

Format

DELETE KEYWORDS *keyword-list-name*

Parameter

keyword-list-name
Specifies the keyword list to be deleted.

Description

The DELETE KEYWORDS command cancels the keyword list defined by the previous DEFINE KEYWORDS command.

Related Commands

DEFINE KEYWORDS
EXTRACT KEYWORDS
SHOW KEYWORDS

Example

```
LSE> DELETE KEYWORDS author_name
```

Cancels the keyword list named *author_name*.

DELETE LANGUAGE

Cancels the indicated language definition.

Format

DELETE LANGUAGE *language-name*

Parameter

language-name
Specifies the language to be deleted.

Description

The DELETE LANGUAGE command cancels the language defined by the previous DEFINE LANGUAGE command. LSE does not actually delete the tokens, placeholders, and aliases associated with the language, but it makes them unavailable for use. If you subsequently issue a DEFINE LANGUAGE command for the same language name, then LSE reassociates all of the previously defined tokens, placeholders, and aliases with the new language definition. Thus, you can use the DELETE LANGUAGE command as a step in modifying the properties of a language definition.

Related Commands

DEFINE LANGUAGE
EXTRACT LANGUAGE
SHOW LANGUAGE

Example

```
LSE> DELETE LANGUAGE ADA
```

Cancels the previously defined characteristics for the Ada language.

DELETE LIBRARY

Deletes an SCA library from a VMS directory.

Format

DELETE LIBRARY *library-spec[,...]*

Qualifiers	**Defaults**
/[NO]CONFIRM	/NOCONFIRM
/[NO]LOG	/NOLOG

Qualifiers

/CONFIRM
/NOCONFIRM (D)
Indicates whether the delete function will request a confirmation of the deletion of each library.

To delete an SCA library, you must respond to the confirmation prompt by typing one or more characters of the word *YES*. Otherwise, the library is not deleted.

/LOG
/NOLOG (D)
Indicates whether successful deletion of SCA libraries will be reported.

Parameter

library-spec[,...]
Specifies one or more libraries to be deleted. The library must be one of the current SCA libraries established by the SET LIBRARY command. You can use a library number in place of a library specification. For example, the primary library is library #1. You can also specify a wildcard name expression.

Related Commands

CREATE LIBRARY
LOAD
SET LIBRARY
SHOW LIBRARY

Example

```
LSE> DELETE LIBRARY/CONFIRM SCA$:[USER.SCA]
```

Deletes a library after confirmation that the library should be deleted.

DELETE MODULE

Deletes specified modules of source analysis data from SCA libraries.

Format

DELETE MODULE *module-name[, . . .]*

Qualifiers	**Defaults**
/[NO]CONFIRM	/NOCONFIRM
/DECLARATION_CLASS= declaration-class	
/LIBRARY[=library-spec]	/LIBRARY=primary-library
/[NO]LOG	/LOG

Qualifiers

/CONFIRM
/NOCONFIRM (D)
Tells SCA whether or not to prompt you to confirm each module deletion.

To delete a module, you must respond to the confirmation prompt by typing Y, YE, or YES. If you specify N, NO, or press the Return key, SCA does not delete the module. SCA considers any other response to be ambiguous and reissues the confirmation prompt.

/DECLARATION_CLASS=declaration-class
Indicates the class of the module to be deleted. The following declaration classes are supported:

PRIMARY—	Module implementation
ASSOCIATED—	Module specification

If you do not specify a declaration class, SCA deletes both classes, if they exist.

/LIBRARY[=library-spec]
/LIBRARY=primary-library (D)
Specifies an SCA library containing the module to be deleted. This library must be one of the current libraries (established by a SET LIBRARY command).

If you do not specify a library, the primary SCA library is the default; that is, the module is deleted from the first of the current SCA libraries.

/LOG (D)
/NOLOG
Indicates whether SCA reports successful deletion of a module.

Parameter

module-name[, . . .]
Specifies the names of the modules to be deleted from the current library. You may specify a wildcard name expression.

Description

The DELETE MODULE command allows you to selectively update a specific SCA library.

Example

```
$ SCA DELETE MODULE module_1
```

Deletes *module_1* from the library.

DELETE PACKAGE

Deletes a package definition without deleting the routines or parameters associated with the package.

Format

DELETE PACKAGE *package-name*

Parameter

package-name
Names the package definition to be deleted.

Description

The DELETE PACKAGE command deletes the specified package. The routines and parameters associated with the package are not deleted, although they are no longer available for use. If a subsequent DEFINE PACKAGE command is issued for the same package name, then all of the previously defined routines and parameters become associated with the new package definition. Thus, you can use the DELETE PACKAGE command, followed by the DEFINE PACKAGE command, to modify the properties of a package definition.

Related Commands

DEFINE PACKAGE
EXTRACT PACKAGE
SHOW PACKAGE

Example

```
LSE> DELETE PACKAGE system_services
```

Deletes the package named *system_services*.

DELETE PARAMETER

Deletes a parameter definition from a package.

Format

DELETE PARAMETER *parameter-name*

Qualifier
/PACKAGE=package-name

Qualifier

/PACKAGE=package-name
Specifies the name of the package that contains the parameter to be deleted. The DELETE PARAMETER command requires this qualifier.

Parameter

parameter-name
Specifies the name of the parameter to be deleted.

Description

The DELETE PARAMETER command deletes the indicated parameter definition from the package specified by the /PACKAGE qualifier.

Related Commands

DEFINE PARAMETER
EXTRACT PARAMETER
SHOW PARAMETER

Example

```
LSE> DELETE PARAMETER/PACKAGE=system_services id
```

Deletes the parameter named *id* from the package named *system_services*.

DELETE PLACEHOLDER

Removes a name from the list of placeholders associated with a language.

Format

DELETE PLACEHOLDER *name*

Qualifier
/LANGUAGE=language-name

Qualifier

/LANGUAGE=language-name
Names the language associated with the placeholder being deleted. By default, LSE deletes the placeholder from the set of placeholders defined for the current language. By using the /LANGUAGE qualifier, you can delete placeholders from other languages as well.

Parameter

name
Specifies the name of the placeholder to be deleted.

Description

The DELETE PLACEHOLDER command removes a specified name from the list of placeholders associated with a language.

DELETE PLACEHOLDER

Related Commands

DEFINE PLACEHOLDER
EXTRACT PLACEHOLDER
SHOW PLACEHOLDER

Example

```
LSE>  DELETE PLACEHOLDER/LANGUAGE=EXAMPLE parameter
```

Removes the placeholder named *parameter* from the list of placeholders associated with the language *EXAMPLE*.

DELETE QUERY

Deletes the indicated query.

Format

DELETE QUERY *[query-name]*

Parameter

query-name
Specifies the query to be deleted. If you omit the query name, the current query is deleted. You can specify wildcards.

Description

The DELETE QUERY command deletes an SCA query.

DECwindows Interface Equivalent

Pop-up menu: Query buffer --> Delete Query

Related Commands

FIND
GOTO QUERY
NEXT QUERY
PREVIOUS QUERY
SHOW QUERY

Example

```
LSE> DELETE QUERY 1
```

Removes the query named *1*.

DELETE ROUTINE

Deletes a routine definition from a package.

Format

DELETE ROUTINE *routine-name*

Qualifier
/PACKAGE=package-name

Qualifier

/PACKAGE=package-name
Indicates the package containing the routine definition to be deleted. The DELETE ROUTINE command requires this qualifier.

Parameter

routine-name
Specifies the name of the routine to be deleted.

Description

The DELETE ROUTINE command deletes a routine definition from a package. If the routine has already been expanded in the current editing session, then the tokens defined by the expansion remain.

Related Commands

DEFINE ROUTINE
EXTRACT ROUTINE
SHOW ROUTINE

Example

```
LSE> DELETE ROUTINE/PACKAGE=system_services sys$add_holder
```

Deletes the routine named *sys$add_holder* from the package named *system_services*.

DELETE TAG

Removes a name from the list of tags associated with a language.

Format

DELETE TAG *name*

Qualifier
/LANGUAGE=language-name

Qualifier

/LANGUAGE=language-name
Names the language associated with the tag being deleted. By default, LSE deletes the tag from the set of tags defined for the current language. By using the /LANGUAGE qualifier, you can delete tags from other languages as well.

Parameter

name
Specifies the name of the tag to be deleted.

Description

The DELETE TAG command removes a specified name from the list of tags associated with a language.

Related Commands

DEFINE TAG
EXTRACT TAG
SHOW TAG

Example

```
LSE>  DELETE TAG/LANGUAGE=EXAMPLE parameters
```

Removes the tag named *parameters* from the list of tags associated with the language *EXAMPLE*.

DELETE TOKEN

Removes a token name from the list of tokens associated with a language.

Format

DELETE TOKEN *name*

Qualifier
/LANGUAGE=language-name

Qualifier

/LANGUAGE=language-name
Specifies the language associated with the token being deleted. By default, LSE deletes the token from the set of tokens defined for the current language. Using the /LANGUAGE qualifier, you can delete tokens from other languages as well.

Parameter

name
Specifies the token name that is to be deleted.

Description

The DELETE TOKEN command removes a token name from the list of tokens associated with either the current language or a language you specify.

Related Commands

DEFINE TOKEN
EXTRACT TOKEN
SHOW TOKEN

Example

```
LSE> DELETE TOKEN/LANGUAGE=EXAMPLE assignment
```

Removes the token *assignment* from the list of tokens associated with the language *EXAMPLE*.

DELETE WINDOW

Deletes the current window.

Format

DELETE WINDOW

Description

The DELETE WINDOW command deletes the current window unless there is only one window. The remaining windows are enlarged to occupy the entire screen.

DECwindows Interface Equivalent

Pull-down menu: Display --> Delete Window

Related Commands

ONE WINDOW
SET SCREEN WINDOW

DO

Directs LSE to execute LSE commands or VAXTPU program statements.

Format

DO *[string[, . . .]]*

Qualifiers	**Defaults**
/BUFFER[=buffer-name]	
/[NO]CONTINUE	/CONTINUE
/LSE	/LSE
/PROMPT=prompt-string	
/TPU	/LSE

Qualifiers

/BUFFER[=buffer-name]
Indicates that LSE should read commands from the specified buffer and execute the commands or VAXTPU program statements within that buffer. The default is the current buffer. If you do not specify either the /BUFFER qualifier or the /PROMPT qualifier, LSE executes the current buffer.

/CONTINUE (D)
/NOCONTINUE
Indicates whether LSE prompts for a single string to be executed, or for multiple strings to be executed. If you specify the /NOCONTINUE qualifier, LSE repeatedly prompts for additional commands until you issue a CONTINUE command.

You use the /[NO]CONTINUE qualifier with the /PROMPT qualifier; you must not specify the /NOCONTINUE qualifier with the /TPU qualifier.

/LSE (D)
Indicates that the strings are LSE commands.

/PROMPT=prompt-string
Indicates that LSE should prompt you for a command (or VAXTPU program statement) to execute.

The /PROMPT and /BUFFER qualifiers are mutually exclusive. If you specify the string parameter, you cannot specify the /PROMPT or /BUFFER qualifier. If you do not specify the /BUFFER qualifier but specify the /PROMPT qualifier, then LSE prompts you for a command and does not execute the current buffer.

/TPU
Indicates that the strings are VAXTPU program statements. When specifying the /TPU qualifier, you cannot use the /NOCONTINUE qualifier.

Parameter

string[, . . .]
Specifies a list of comma-separated commands or statements to be executed. Commands with embedded spaces, such as GOTO BUFFER, must be enclosed by double quotation marks.

Description

With the DO command, you can issue commands from a command line or from a buffer. You may specify a list of commands to be executed, or direct LSE to prompt you for LSE/SCA commands or VAXTPU program statements (see the *VAX Text Processing Utility Reference Manual* for a description of VAXTPU programs).

To end the prompting for commands and return to keypad editing, you issue the CONTINUE command or press CTRL/Z.

Keypad Equivalent

DO/CONTINUE/PROMPT=LSE Command>

Key	Keypad Mode
PF1-KP7 COMMAND	EDT LK201, EDT VT100, EVE LK201
DO DO	EDT LK201, EVE LK201
PF4 DO	EVE VT100

DO/NOCONTINUE/PROMPT=LSE>

Key	Keypad Mode
CTRL/Z	All

DO/CONTINUE/TPU/PROMPT=TPU>

Key	Keypad Mode
PF1-CTRL/Z	All

Related Commands

CALL
CONTINUE
GOTO COMMAND
EXTEND

Examples

1.
```
LSE> DO "GOTO LINE","PASTE"
```

Moves the cursor to the end of the line in the current direction and copies the contents of the $PASTE buffer at that position.

2. `LSE> DO/TPU "ERASE(MESSAGE_BUFFER)"`

   Invokes VAXTPU to erase the contents of the message buffer. Any messages that have accumulated at the bottom of your screen now disappear.

END DEFINE

Ends a body of text that begins with a DEFINE command.

Format

END DEFINE

Description

The END DEFINE command ends the body that follows a DEFINE PLACEHOLDER or DEFINE TOKEN command if the placeholder or token definition has a body. The END DEFINE command ends the list of keywords defined by the DEFINE KEYWORDS command.

The END DEFINE command also ends the sequence of keystrokes that follows a DEFINE KEY/LEARN command. To use the END DEFINE command for this purpose, you must issue the command by pressing a key you have defined to be the END DEFINE key.

Keypad Equivalent

Key	Keypad Mode
CTRL/R	EVE LK201, EVE VT100
None	EDT LK201, EDT VT100

Related Commands

DEFINE KEY
DEFINE KEYWORDS
DEFINE PLACEHOLDER
DEFINE TOKEN

Example

```
DEFINE PLACEHOLDER parameter -
  /LANGUAGE = EXAMPLE -
  /DESCRIPTION = "Parameter name"
  /DUPLICATION = HORIZONTAL -
  /SEPARATOR = ", " -
  /TYPE = TERMINAL
  "A string of letters and digits starting with a letter."
END DEFINE
```

Shows the position of the END DEFINE command at the end of a DEFINE PLACEHOLDER command.

END REVIEW

Ends an LSE REVIEW session.

Format

END REVIEW

Description

The END REVIEW command ends the current REVIEW session (initiated by a REVIEW or COMPILE/REVIEW command) and deletes the window containing the $REVIEW buffer.

DECwindows Interface Equivalent

Pop-up menu: Review buffer --> End Review

Related Commands

REVIEW

ENLARGE WINDOW

Enlarges the current window.

Format

ENLARGE WINDOW *line-count*

Parameter

line-count
Specifies the number of screen lines you want to add to the current window. If you do not supply this parameter, LSE prompts you for the number of lines to add.

The maximum size of a window depends on the size and type of the terminal screen you are using. The minimum size is one line of text and one line for the status line.

Description

The ENLARGE WINDOW command enlarges the window that the text cursor is in (if you are using more than one window). LSE shrinks the other window (or windows) accordingly.

Related Commands

SHRINK WINDOW

Example

```
LSE>  ENLARGE WINDOW 10
```

Adds ten lines to the current window, taking them proportionally from the other window (or windows) on the screen.

ENTER COMMENT

Converts pseudocode into comments.

Format

ENTER COMMENT

Qualifiers	**Defaults**
/BLOCK	/BLOCK
/LINE	/BLOCK

Qualifiers

/BLOCK (D)
Specifies that the comment should be entered above the cursor (or selected text range), formatting the comment according to the placeholder LSE$BLOCK_COMMENT.

You cannot specify both the /BLOCK and /LINE qualifiers.

/LINE
Specifies that the comment should be entered at the end of the current line (or selected text range), formatting the comment according to the placeholder LSE$LINE_COMMENT.

You cannot specify both the /LINE and /BLOCK qualifiers.

Description

The ENTER COMMENT command converts pseudocode into comments. It inserts a comment near the current cursor position.

If the cursor is on a pseudocode placeholder, the command moves the placeholder's text into the comment and replaces the placeholder with the LSE$GENERIC placeholder. The cursor is then positioned on the generic placeholder.

If the cursor is in a comment, then the LSE editor finds a nearby pseudocode placeholder P and converts P's content into a comment. The LSE$GENERIC placeholder is inserted in place of P and the cursor remains on the generic placeholder. The command qualifiers are ignored when the cursor is on a comment.

If the cursor is not on a placeholder or a comment, then the command inserts a new comment and puts the LSE$GENERIC placeholder inside the comment. The cursor is then positioned on the generic placeholder.

If there is a sequence of pseudocode placeholders and a selected range is active when ENTER COMMENT is executed, all text in the selected range is converted into a comment and the placeholders are replaced with LSE$GENERIC placeholders accordingly. The cursor is positioned on the first placeholder after the comment.

The ENTER COMMENT command requires definitions for three placeholders as follows:

LSE$BLOCK_COMMENT	Specifies the comment format to be used by ENTER COMMENT/BLOCK.
LSE$LINE_COMMENT	Specifies the comment format to be used by ENTER COMMENT/LINE.
LSE$GENERIC	Specifies the text to be inserted in place of the pseudocode placeholder removed by ENTER COMMENT.

The following are example definitions for Ada:

```
DEFINE PLACEHOLDER LSE$BLOCK_COMMENT /TYPE=NOTERMINAL
    "-- {tbs}"
    "--"
END DEFINE

DEFINE PLACEHOLDER LSE$LINE_COMMENT /TYPE=NOTERMINAL
    "-- {tbs}"
END DEFINE

DEFINE PLACEHOLDER LSE$GENERIC /TYPE=NOTERMINAL
    "{tbs}"
END DEFINE
```

The following is an example definition for COBOL:

```
DEFINE PLACEHOLDER LSE$BLOCK_COMMENT
    "*"/INDENTATION=(fixed,1), "{tbs}"
    "*"/INDENTATION=(fixed,1)
```

The following is an example definition for FORTRAN:

```
DEFINE PLACEHOLDER LSE$BLOCK_COMMENT
    "!", " {tbs}"/INDENTATION=EXPAND
    "!"
```

Keypad Equivalent

ENTER COMMENT/BLOCK

Key	Keypad Mode
PF1-B	All

ENTER COMMENT/LINE

Key	Keypad Mode
PF1-L	All

Related Commands

UNDO ENTER COMMENT

Examples

The following are examples of converting pseudocode to comments:

1. `«This is something interesting.»`

   Issuing the ENTER COMMENT/LINE command causes LSE to convert the pseudocode placeholder to a comment as follows:

   ```
   {tbs} ! This is something interesting.
   ```

2.
```
«We will move the third item from the left to be the»
«next to the last item from the right in this case.»
```

   If there is a selected range active for both lines, issuing the ENTER COMMENT/BLOCK command causes LSE to convert pseudocode to comments, as follows:

```
-- We will move the third item from the left to be the
-- next to the last item from the right in this case.
{tbs}
```

ENTER LINE

Splits the current line into two lines.

Format

ENTER LINE

Qualifiers	**Defaults**
/BEGINNING	/BEGINNING
/[NO]COMMENT	/COMMENT
/END	/BEGINNING

Qualifiers

/BEGINNING (D)
Indicates that the cursor should be left at the beginning of the second line. If you position the cursor at the *end* of the original line, the /BEGINNING qualifier adds a new blank line to the current buffer and repositions the cursor at the beginning of the new line.

If you position the cursor at the *beginning* of a line, the /BEGINNING qualifier adds a new blank line before the current line and the cursor remains at the beginning of the current line.

If you position the cursor *within* a line, the /BEGINNING qualifier splits that line into two lines at the original cursor position and repositions the cursor at the beginning of the second line.

/COMMENT (D)
/NOCOMMENT
Indicates whether or not the second line should be a comment. This qualifier has no effect unless each of the following conditions are met:

- The current buffer is associated with a language
- Comments are defined for the language

- The cursor is positioned within a comment
- Wrapping is set for the current buffer

If all these conditions apply, you use the /NOCOMMENT qualifier when you want to terminate a comment and begin a code line.

/END
Indicates that the cursor should be left at the end of the first line. If you start with the cursor at the *end* of the original line, the /END qualifier causes the cursor to stay there.

If you start with the cursor at the *beginning* of a line, the /END qualifier adds a new blank line before the current line and positions the cursor on that blank line.

If you position the cursor *within* a line, specifying the /END qualifier splits the line in two leaving the cursor at the end of the first line.

Description

The ENTER LINE command splits the current line into two lines and places the cursor at the end of the first line or the beginning of the second line, depending on the qualifier you specify.

The ENTER LINE command also works in conjunction with the SET WRAP command to let you fill lines of text between margins. If wrapping is set for the buffer, then LSE indents the second line to the left margin.

Keypad Equivalent

ENTER LINE/BEGINNING

Key	Keypad Mode
Return	All

ENTER LINE/END

Key	Keypad Mode
PF1-KP0 OPEN LINE	EDT LK201, EDT VT100, EVE LK201

ENTER LINE/NOCOMMENT

Key	Keypad Mode
PF1-Return	All

Related Commands

ENTER SPACE
SET WRAP

ENTER PSEUDOCODE

Inserts pseudocode placeholder delimiters.

Format

ENTER PSEUDOCODE

Description

The ENTER PSEUDOCODE command inserts pseudocode placeholder delimiters and positions the cursor on the first character of the right delimiter. The pseudocode placeholder delimiters must be defined before using this command.

If the cursor is on a placeholder defined with the command DEFINE PLACEHOLDER/PSEUDOCODE, then the command has the usual effects of text insertion on the defined placeholders. The defined placeholder is autoerased and, if it is a list placeholder, it is duplicated.

If the cursor is on a placeholder defined with the command DEFINE PLACEHOLDER/NOPSEUDOCODE, or is on a pseudocode placeholder, then the command is not allowed and a warning message is displayed.

If the cursor is not on a placeholder, the command inserts the pseudocode placeholder delimiter.

Keypad Equivalent

Key	Keypad Mode
PF1-Space bar	ALL

Related Commands

DEFINE LANGUAGE
DEFINE PLACEHOLDER
MODIFY LANGUAGE

Examples

The following are examples of entering pseudocode:

1.
```
IF {expression}
```

Issuing the ENTER PSEUDOCODE command causes LSE to insert pseudocode placeholder delimiters as follows:

```
IF «»
```

2.
```
{statement} . . .
```

Issuing the ENTER PSEUDOCODE command causes LSE to insert pseudocode placeholder delimiters as follows:

```
«»
[statement] . . .
```

ENTER SPACE

Inserts or overstrikes a space at the current cursor position, depending on whether the current editing mode is insert or overstrike.

If wrap mode is set, then line-oriented filling occurs.

Format

ENTER SPACE

Description

The ENTER SPACE command either inserts or overstrikes a space depending on the current editing mode. If the cursor is past the right margin and wrap mode is set, the ENTER SPACE command performs a line-fill operation on the current line (see the SET [NO]WRAP command). You can change the right margin with the SET RIGHT_MARGIN command.

Keypad Equivalent

Key	Keypad Mode
Space bar	All

Related Commands

ENTER LINE
SET WRAP

ENTER SPECIAL

Causes LSE to insert into the current buffer a character whose ASCII code you specify.

Format

ENTER SPECIAL *ASCII-code*

Parameter

ASCII-code
Specifies the ASCII code of the character you want as a decimal number from 0 through 255.

Description

The ENTER SPECIAL command inserts a special character into the buffer at the current cursor position. You can insert a form feed or other nonprinting characters as well as printing characters, such as letters and punctuation marks. When you issue the command, LSE prompts you for the ASCII code of the character you want to insert.

Keypad Equivalent

ENTER SPECIAL

Key	Keypad Mode
PF1-KP3 SPECINS	EDT LK201, EDT VT100, EVE LK201
CTRL/V	All

Related Commands

QUOTE

Example

```
LSE>  ENTER SPECIAL 12
```

Causes LSE to insert a form-feed character (CTRL/L).

ENTER TAB

Inserts tabs and blanks at the current cursor position.

Format

ENTER TAB

Description

The ENTER TAB command inserts tabs and blanks at the current cursor position. If the cursor is at the beginning of the line, LSE inserts tabs and blanks up to the current indentation level. If the current indentation level is set at the beginning of the line, the ENTER TAB command does not insert tabs and blanks. If the cursor is not at the beginning of the line, the ENTER TAB command inserts an ASCII tab character.

Related Commands

SET INDENTATION
SET TAB_INCREMENT
TAB

ENTER TEXT

Inserts text at the current cursor position.

Format

ENTER TEXT *string*

Parameter

string
Is a quoted string specifying the text to be inserted.

Description

The ENTER TEXT command inserts text from a quoted string at the current cursor position.

Example

```
LSE> ENTER TEXT "Insert this"
```

Inserts the quoted text at the current cursor position.

ERASE CHARACTER

Erases a single character at the current cursor position.

Format

ERASE CHARACTER

Qualifiers	Defaults
/CURRENT	/CURRENT
/FORWARD	/CURRENT
/INDICATED	/INDICATED
/REVERSE	/CURRENT
/TO	/INDICATED

Qualifiers

/CURRENT (D)
Erases text in the current direction.

/FORWARD
Erases text in the forward direction.

/INDICATED (D)
Deletes the character at the current cursor position.

/REVERSE
Erases text in the reverse direction.

/TO
Deletes the character at the current cursor position when the direction is FORWARD. Deletes the character before the current cursor position when the direction is REVERSE.

ERASE CHARACTER

Description

The ERASE CHARACTER command removes a single character from the current buffer. (A line terminator or ASCII tab character is considered one character.) In either insert or overstrike mode, the remainder of the line moves left one character to close up the space. An exception is the ERASE/TO CHARACTER/REVERSE command, which in overstrike mode changes the erased character to a space and moves left one position.

When the cursor is at the end of a line, the carriage return is deleted, and the text from the following line moves up to the right of the text in the current line.

Keypad Equivalent

ERASE/TO CHARACTER/REVERSE

Key	Keypad Mode
Delete	All

ERASE/TO CHARACTER/FORWARD

Key	Keypad Mode
Keypad comma (,) DEL C	EDT LK201, EDT VT100, EVE LK201
None	EVE VT100

Related Commands

UNERASE CHARACTER

Example

```
LSE> ERASE CHARACTER
```

Deletes the character at the current cursor position (equivalent to pressing the comma key on the EDT numeric keypad).

ERASE LINE

Removes a line of text at the current cursor position.

Format

ERASE LINE

Qualifiers	Defaults
/BEGINNING	/BEGINNING
/CURRENT	/CURRENT
/END	/BEGINNING
/FORWARD	/CURRENT
/INDICATED	/INDICATED
/REVERSE	/CURRENT
/TO	/INDICATED

Qualifiers

/BEGINNING (D)
Indicates that the cursor should be moved to the beginning of a line as part of the ERASE operation. You cannot use the /BEGINNING qualifier with the /INDICATED qualifier.

/CURRENT (D)
Erases text in the current direction.

/END
Indicates that the cursor should be moved to the end of a line as part of the ERASE operation. You cannot use the /END qualifier with the /INDICATED qualifier.

/FORWARD
Erases text in the forward direction.

/INDICATED (D)
Erases the entire line that the cursor is on (including the carriage return and line feed), regardless of the cursor position within that line or the direction specified. The cursor moves to the beginning of the next line. You cannot use the /INDICATED qualifier with the /BEGINNING, /END, or /TO qualifiers.

/REVERSE
Erases text in the reverse direction.

/TO
Erases text from the current cursor position to the next line in the direction specified.

Description

The ERASE LINE command removes a line of text from the current cursor position. When LSE deletes all of the text from the current cursor position to the end of the current line, the text on the following line moves up to fill the space to the right of the cursor.

Keypad Equivalent

ERASE/TO LINE/BEGINNING/REVERSE

Key	Keypad Mode
CTRL/U	All

ERASE/TO LINE/BEGINNING/FORWARD

Key	Keypad Mode
PF4 DEL L	EDT LK201, EDT VT100, EVE LK201
None	EVE VT100

ERASE LINE

ERASE/TO LINE/END

Key	Keypad Mode
PF1-KP2 DEL EOL	EDT LK201, EDT VT100, EVE LK201
None	EVE VT100

Related Commands

UNERASE LINE

Examples

1.
   ```
LSE> ERASE LINE
```

   Erases the entire line that the cursor is on, regardless of cursor position or direction specified.

2. CTRL/U

   Erases text from the current cursor position to the beginning of the current line. If the cursor is already at the beginning of a line, CTRL/U erases to the beginning of the previous line.

3.
   ```
LSE> ERASE/TO LINE/END
```

   Erases text from the current cursor position to the end of the current line, but does not erase the line break.

ERASE PLACEHOLDER

Deletes the text of a placeholder and related punctuation.

Format

ERASE PLACEHOLDER

Qualifiers	**Defaults**
/CURRENT	/CURRENT
/FORWARD	/CURRENT
/[NO]GOTO_PLACEHOLDER	/GOTO_PLACEHOLDER
/REVERSE	/CURRENT

Qualifiers

/CURRENT (D)
Specifies cursor motion in the current direction.

/FORWARD
Specifies cursor motion in the forward direction.

/GOTO_PLACEHOLDER (D)
/NOGOTO_PLACEHOLDER
Specifies whether or not the cursor should move to the next placeholder after performing the ERASE operation. The movement to the next placeholder does not take place if it would force the current position to scroll off the screen.

/REVERSE
Specifies cursor motion in the reverse direction.

ERASE PLACEHOLDER

Description

The ERASE PLACEHOLDER command moves the cursor to the next placeholder in the direction specified and deletes the placeholder. The implicit GOTO PLACEHOLDER command caused by the ERASE PLACEHOLDER command goes only to regular LSE placeholders, not to pseudocode placeholders. If the cursor is already on a placeholder, then the deletion occurs in place.

If the cursor is on a character of a closing pseudocode placeholder delimiter, or not on a placeholder, then the ERASE PLACEHOLDER command performs a GOTO PLACEHOLDER command before erasing.

If no placeholder is found, LSE returns an error message.

After deleting the placeholder and any leading tabs or blanks, LSE then deletes any leading separator text or leading and trailing punctuation. If the resulting line or line segment is now empty, LSE then deletes the entire line or line segment.

Keypad Equivalent

ERASE PLACEHOLDER/FORWARD

Key	Keypad Mode
CTRL/K	All

Related Commands

DEFINE PLACEHOLDER
UNERASE PLACEHOLDER

Examples

```
DEFINE PLACEHOLDER identifier_list -
    /TRAILING=":" -
    /SEPARATOR=","
              . . .
```

This DEFINE PLACEHOLDER specification applies to each of the following examples. The line comment delimiter is a double hyphen (--).

1. `<text> [identifier_list] <more text>`

   If this is the original text, then issuing an ERASE PLACEHOLDER command produces the following:

   ```
    <text> <more text>
   ```

2. `<text> [identifier_list] :  <more text>`

   If this is the original text, then issuing an ERASE PLACEHOLDER command produces the following:

   ```
    <text> <more text>
   ```

3. `<text> , [identifier_list] <more text>`

   If this is the original text, then issuing an ERASE PLACEHOLDER command produces the following:

   ```
    <text> <more text>
   ```

4. `<text> -- [identifier_list] <more text>`

   If this is the original text, then issuing an ERASE PLACEHOLDER command produces the following:

   ```
    <text> -- <more text>
   ```

5.
   ```
           -- [identifier_list] <more text>
   ```

   If this is the original text, then issuing an ERASE PLACEHOLDER command produces the following:

   ```
           -- <more text>
   ```

ERASE SELECTION

Removes the text within the selected range.

Format

ERASE SELECTION

Description

The ERASE SELECTION command removes the text within the selected range. The selected range is the text between the select marker (see the SET SELECT_MARK command) and the current cursor position.

DECwindows Interface Equivalent

ERASE SELECTION
Pull-down menu: Edit --> Clear

Related Commands

UNERASE SELECTION

Example

```
LSE>   ERASE SELECTION
```

Removes the text within the selected range.

ERASE WORD

Removes a word at the current cursor position.

Format

ERASE WORD

Qualifiers	Defaults
/CURRENT	/CURRENT
/FORWARD	/CURRENT
/INDICATED	/INDICATED
/NEXT	
/PREVIOUS	
/REVERSE	/CURRENT
/TO	/INDICATED

Qualifiers

/CURRENT (D)
Erases text in the current direction.

/FORWARD
Erases text in the forward direction.

/INDICATED (D)
Deletes the entire word the cursor is on, regardless of the cursor's position within that word.

/NEXT
Erases the word following the cursor. When the cursor is positioned on a space, LSE erases all the spaces before and after the deleted word except one space. If the cursor is at the end of a line, the next line is appended to the current line. You cannot use the /NEXT qualifier with any other ERASE WORD qualifier.

/PREVIOUS
Erases the previous word when the cursor is on the first character of a word or between words. When the cursor is in the middle of a word, that entire word is erased and the cursor moves on to the first letter of the next word. You cannot use the /PREVIOUS qualifier with any other ERASE WORD qualifier.

/REVERSE
Erases text in the reverse direction.

/TO
Deletes text from the current cursor position to the beginning of the next word in the specified direction.

Description

The ERASE WORD command removes a word from the current buffer. A word can be terminated by tabs or characters not specified in the /IDENTIFIER_CHARACTERS qualifier on the DEFINE LANGUAGE command. A word can consist of identifier characters and trailing blanks, or it can consist of a single nonblank, nonidentifier character.

Keypad Equivalent

ERASE/TO WORD/REVERSE

Key	Keypad Mode
F13 DEL PRV W	EDT LK201
CTRL/J LINEFEED	All

ERASE/TO WORD/FORWARD

Key	Keypad Mode
Keypad minus (–) DEL W	EDT LK201, EDT VT100, EVE LK201
None	EVE VT100

ERASE WORD/NEXT

Key	Keypad Mode
Keypad comma (,) ERASE WORD	EVE VT100
F13 ERASE WORD	EVE LK201

Related Commands

UNERASE WORD

Example

```
LSE> ERASE WORD
```

Deletes the entire word at the current cursor position.

EXIT

Ends an LSE editing session or SCA query session and returns control to the calling process or the VMS command language interpreter.

Format

EXIT

Qualifier	**Default**
/[NO]LOG {SCA only}	/NOLOG

Qualifier

/LOG
/NOLOG (D)
Indicates whether completion of an SCA session is reported.

Description

The EXIT command ends or suspends your session and returns control to the process that called LSE or SCA (usually the DCL command interpreter). If you are using LSE, the contents of buffers that are associated with files are written to their files if they have been modified. Buffers with the READ_ONLY attribute are not written back.

Keypad Equivalent

Key	**Keypad Mode**
F10 EXIT	EDT LK201, EVE LK201
None	EDT VT100, EVE VT100

DECwindows Interface Equivalent

Pull-down menu: File --> Exit

Related Commands

ATTACH
QUIT
SPAWN

Examples

1.
```
LSE> EXIT
```

Ends an LSE session and writes modified buffers back to their respective files.

2.
```
SCA> EXIT
```

Ends an SCA query session.

EXPAND

Replaces placeholders, token names, alias names, or routine names at the current cursor position with the appropriate body of text or code, if the cursor is not on the overview line. Replaces overview line with the underlying source lines, if the cursor is on the overview line. Expands symbols to include their occurrences, if the cursor is in a query buffer.

Format

EXPAND

Qualifiers	**Defaults**
/DEPTH=*n*	/DEPTH=1
/[NO]GOTO_PLACEHOLDER	/GOTO_PLACEHOLDER

Qualifiers

/DEPTH=n
/DEPTH=1 (D)
Specifies how many levels of detail are displayed. If you specify the value ALL, all subgroups for this overview line are expanded.

If the cursor is not on an overview line or is in a query buffer, the /DEPTH qualifier is ignored. Note that when you use the EXPAND command with SCA, this command does not support the /DEPTH qualifier.

/GOTO_PLACEHOLDER (D)
/NOGOTO_PLACEHOLDER
Specifies whether or not the cursor should move to the next placeholder after performing the EXPAND operation. The movement to the next placeholder does not take place if it would force the current position to scroll off the screen.

If the cursor is on an overview line, the /GOTO_PLACEHOLDER qualifier is ignored. Note that when you use the EXPAND command with SCA, this command does not support the /GOTO_PLACEHOLDER qualifier.

Description

If the cursor is not on an overview line, the EXPAND command expands text representing alias names, routine names, token names, or placeholders at the current position.

The EXPAND_CASE setting (defined with the DEFINE LANGUAGE command or MODIFY LANGUAGE command) determines the case of the inserted text. If the EXPAND_CASE is UPPER or LOWER, LSE inserts the text in that case. If the EXPAND_CASE is AS_IS, LSE inserts the text as it appears in the token definition.

If the cursor is on an overview line, the overview is expanded to display the underlying hidden text.

The editor determines the relative level of detail of a line by comparing the indentation of the line with the indentation of other lines. The editor's treatment of the indentation of a line is influenced by indentation adjustment definitions. For more information, see the DEFINE ADJUSTMENT command.

For SCA, if the cursor is positioned on a symbol in a query buffer, the EXPAND command expands the symbol to display its occurrences.

Keypad Equivalent

EXPAND

Key	Keypad Mode
CTRL/E	EDT LK201, EDT VT100
CTRL//	EVE LK201, EVE VT100

EXPAND/DEPTH=ALL

Key	Keypad Mode
PF1-<	All

EXPAND

DECwindows Interface Equivalent

EXPAND
Pop-up menu: Query buffer --> Expand
Pull-down menu: View --> Expand

EXPAND/DEPTH=ALL
Pull-down menu: View --> Expand All

Related Commands

COLLAPSE
DEFINE ADJUSTMENT
DEFINE LANGUAGE/OVERVIEW_OPTIONS
FOCUS
MODIFY LANGUAGE
SET NOOVERVIEW
SET OVERVIEW
UNEXPAND
VIEW SOURCE

Examples

The following are examples of replacing a token or nonterminal placeholder with its body text based on the token or placeholder definition.

1.

```
DEFINE TOKEN for -
    /LANGUAGE=C -

    "for ([@expression@];  [@expression@];  [@expression@])"
    "{@statement@}"/INDENTATION=(EXPAND,1,TAB)

END DEFINE

DEFINE TOKEN "{" -
    /LANGUAGE=C -

    "{"/INDENTATION=PREVIOUS
    "{@statement@}..."/INDENTATION=(PREVIOUS, 1, TAB)
    "}"/INDENTATION=PREVIOUS

END DEFINE
```

With the definitions in this example, typing "{" on the placeholder {@statement@} (Step 1) and expanding it (Step 2) produces the following (Step 3):

```
Step 1:
    for ( i = 0; i >15; i++ )
        {@statement@}
Step 2:
    for ( i = 0; i >15; i++ )
        {
Step 3:
    for ( i = 0; i >15; i++ )
    {
        {@statement@}...
    }
```

2.
```
DEFINE PLACEHOLDER "#IF" -
    /LANGUAGE= C -

    "#if {@constant expression@}"/INDENTATION=(FIXED,1)
    "[@#else_clause@]"/INDENTATION=(FIXED,1)
    "#endif"/INDENTATION=(FIXED,1)

END DEFINE
```

With the definitions in this example, expanding the [@#IF@] placeholder at any column always yields indentation to the column defined, as follows:

```
Step1:
        [@#if@]

Step2:

#if {@constant expression@}
[@#else_clause@]
#endif
```

EXTEND

Compiles one or more VAXTPU procedures to extend LSE.

Format

$$\textbf{EXTEND} \quad \left\{ \begin{array}{l} \textit{procedure-name} \\ * \end{array} \right\}$$

Qualifier	**Default**
/INDICATED	/INDICATED

Qualifier

/INDICATED (D)
If you specify the /INDICATED qualifier, the EXTEND command compiles the VAXTPU procedure in which the cursor is located. You cannot specify the /INDICATED qualifier with a parameter.

Parameters

procedure-name
The name of the VAXTPU procedure you want to compile. You can abbreviate the procedure name.

*

Wildcard symbol instructing VAXTPU to compile all the procedures and statements in the buffer.

Description

The EXTEND command compiles one or more VAXTPU procedures to extend LSE. Using EXTEND without specifying the procedure name compiles the procedure in which the cursor is located.

To execute a compiled procedure, use the EXTEND command followed by the name of the procedure you want executed. To save a compiled procedure in a section file for future editing sessions, use the SAVE SECTION command.

If the procedure contains any overview records, a message informs you that the operation cannot be performed because there are overview records in the selected range. Compiler messages appear in the message window.

You cannot specify a parameter with the /INDICATED qualifier.

Example

```
LSE>  EXTEND user_proc
```

Compiles a procedure called *USER_PROC*.

EXTRACT ADJUSTMENT

Extracts the definition of the named adjustment and formats the definition as a command.

Format

EXTRACT ADJUSTMENT *adjustment-name*

Qualifiers
/LANGUAGE[=language-name]
/NEW

Qualifiers

/LANGUAGE[=language-name]
Specifies the language associated with the adjustment being extracted. If you do not specify the /LANGUAGE qualifier, the default is the current language.

/NEW
Specifies that only the adjustment definitions defined during this editing session should be extracted.

Parameter

adjustment-name
Specifies the name of the adjustment you want. You may specify a wildcard.

Description

The EXTRACT ADJUSTMENT command extracts the named adjustment definition and formats it as a command. LSE inserts the indicated definitions at the end of the current buffer in a form that permits them to be read back and to replace existing definitions. Specifically, the DELETE ADJUSTMENT command precedes the corresponding DEFINE ADJUSTMENT command.

With the EXTRACT ADJUSTMENT command, you can modify definitions by editing and then executing them using the DO command. You can write definitions to a file.

You can use this command to extract adjustments to make global changes to them. Once you have edited the buffer, use the DO command to execute the changes.

You can create new definitions in a buffer, and edit and execute them until they are correct.

Related Commands

DEFINE ADJUSTMENT
DELETE ADJUSTMENT
SHOW ADJUSTMENT

Example

```
LSE> EXTRACT ADJUSTMENT/LANGUAGE=ADA then
```

Extracts the current definition of the *then* adjustment from the list of adjustments associated with the Ada language and places the definition at the end of the current buffer.

EXTRACT ALIAS

Extracts the definition of an alias and formats the definition as a command.

Format

EXTRACT ALIAS *alias-name*

Qualifiers
/LANGUAGE[=language-name]
/NEW

Qualifiers

/LANGUAGE[=language-name]
Specifies the language associated with the alias being extracted. If you do not specify the /LANGUAGE qualifier, the default is the current language.

/NEW
Specifies that only the definitions of aliases defined during this editing session should be extracted.

Parameter

alias-name
Specifies the name of the alias you want. You may specify a wildcard.

Description

The EXTRACT ALIAS command extracts the named alias definition and formats it as a command. LSE inserts the indicated definitions at the end of the current buffer in a form that permits them to be read back and to replace existing definitions. Specifically, the DELETE ALIAS command precedes the corresponding DEFINE ALIAS command.

With the EXTRACT ALIAS command, you can modify alias definitions by editing and then executing them using the DO command. You can write definitions to a file.

You can create new definitions in a buffer, and edit and execute them until they are correct.

Related Commands

DEFINE ALIAS
DELETE ALIAS
SHOW ALIAS

Example

```
LSE> EXTRACT ALIAS EXE
```

Places the current definition of the *EXE* alias at the end of the current buffer.

EXTRACT KEYWORDS

Extracts the definition of the named keyword list and formats the definition as a command.

Format

EXTRACT KEYWORDS *keyword-list-name*

Qualifier
/NEW

Qualifier

/NEW
Specifies that only the definitions of keyword list names defined during this editing session should be extracted.

Parameter

keyword-list-name
Specifies the keyword list name. You can specify a wildcard.

Description

The EXTRACT KEYWORDS command extracts the named keyword list definition and formats it as a command. LSE inserts the indicated definitions at the end of the current buffer in a form that permits them to be read back and to replace existing definitions. Specifically, the DELETE KEYWORDS command precedes the corresponding DEFINE KEYWORDS command.

With the EXTRACT KEYWORDS command, you can modify keyword list definitions by editing and then executing them using the DO command. You can write definitions to a file.

You can create new definitions in a buffer, and edit and execute them until they are correct.

Related Commands

DEFINE KEYWORDS
DELETE KEYWORDS
SHOW KEYWORDS

Example

```
LSE> EXTRACT KEYWORDS author_name
```

Places the current definition of the keyword list *author_name* at the end of the current buffer.

EXTRACT LANGUAGE

Extracts the definition of the named language and formats the definition as a command.

Format

EXTRACT LANGUAGE *language-name*

Qualifier
/NEW

Qualifier

/NEW
Specifies that only the definitions of languages defined during this editing session should be extracted.

Parameter

language-name
Specifies the name of the language you want. You may specify a wildcard.

Description

The EXTRACT LANGUAGE command extracts the named language definition and formats it as a command. LSE inserts the indicated definitions at the end of the current buffer in a form that permits them to be read back and to replace existing definitions. Specifically, the DELETE LANGUAGE command precedes the corresponding DEFINE LANGUAGE commands.

With the EXTRACT LANGUAGE command, you can modify language definitions by editing and then executing them using the DO command. You can write definitions to a file.

You can create new definitions in a buffer, and edit and execute them until they are correct.

Related Commands

DEFINE LANGUAGE
DELETE LANGUAGE
SHOW LANGUAGE

Example

```
LSE> EXTRACT LANGUAGE Pascal
```

Places the current definition of the *Pascal* language at the end of the current buffer.

EXTRACT MODULE

Extracts specified modules of source analysis data from an SCA library.

Format

EXTRACT MODULE *module-name-expr[, . . .]*

Qualifiers	**Default**
/DECLARATION_CLASS=declaration-class	
/LIBRARY=library-spec	
/[NO]LOG	/NOLOG
/OUTPUT=file-spec	

Qualifiers

/DECLARATION_CLASS=declaration-class
Indicates the class of the module to be copied. The following declaration classes are supported:

PRIMARY—Module implementation
ASSOCIATED—Module specification

If you do not specify a declaration class, SCA extracts both classes, if they exist.

/LIBRARY=library-spec
Specifies the SCA static library from which to extract the module. This library must be one of the current SCA libraries (established by a SET LIBRARY command). If you do not specify this qualifier, then SCA tries to extract the module from the primary library (the first of the current SCA libraries).

/LOG
/NOLOG (D)
Indicates whether SCA reports the extraction of a module.

/OUTPUT=file-spec
Specifies the file into which all modules of source analysis data will be written. The default is /OUTPUT=module-name.ANA, where the module name is the name of the file the compiler created.

Parameter

module-name-expr[, . . .]
Specifies the modules to extract. If you specify more than one library, SCA extracts the module from the first library in which it occurs.

Description

The EXTRACT MODULE command extracts the named module from the specified SCA static library and places it in a file of type .ANA, which is the file type for source analysis data files created by compilers. The EXTRACT MODULE command performs the reverse function of the LOAD command.

Related Commands

LOAD
SET LIBRARY

Example

```
$  SCA EXTRACT MODULE module_1
```

Extracts *module_1* from the current library.

EXTRACT PACKAGE

Extracts the definition of the named package and formats the definition as a command.

Format

EXTRACT PACKAGE *package-name*

Qualifiers
/LANGUAGE[=language-name]
/NEW

Qualifiers

/LANGUAGE[=language-name]
Specifies the language associated with the package being extracted. If you do not specify the /LANGUAGE qualifier, the default is the current language.

/NEW
Specifies that only the definitions of packages defined during this editing session should be extracted.

Parameter

package-name
Specifies the name of the package you want. You may specify a wildcard.

Description

The EXTRACT PACKAGE command extracts the named package definition and formats it as a command. LSE inserts the indicated definitions at the end of the current buffer in a form that permits them to be read back and to replace existing definitions. Specifically, the DELETE PACKAGE command precedes the corresponding DEFINE PACKAGE command.

With the EXTRACT PACKAGE command, you can modify package definitions by editing and then executing them using the DO command. You can write definitions to a file.

You can create new definitions in a buffer, and edit and execute them until they are correct.

Related Commands

DEFINE PACKAGE
DELETE PACKAGE
SHOW PACKAGE

Example

```
LSE>  EXTRACT PACKAGE system_services
```

Places the current definition of the *system_services* package at the end of the current buffer.

EXTRACT PARAMETER

Extracts the definition of the named parameter and formats the definition as a command.

Format

EXTRACT PARAMETER *parameter-name*

Qualifiers
/LANGUAGE[=language-name]
/NEW

Qualifiers

/LANGUAGE[=language-name]
Specifies the language associated with the parameter being extracted. If you do not specify the /LANGUAGE qualifier, the default is the current language.

/NEW
Specifies that only the definitions of parameters defined during this editing session should be extracted.

Parameter

parameter-name
Specifies the name of the parameter you want. You may specify a wildcard.

Description

The EXTRACT PARAMETER command extracts the named parameter definition and formats it as a command. LSE inserts the indicated definitions at the end of the current buffer in a form that permits them to be read back and to replace existing definitions. Specifically, the DELETE PARAMETER command precedes the corresponding DEFINE PARAMETER command.

With the EXTRACT PARAMETER command, you can modify definitions by editing and then executing them using the DO command. You can write definitions to a file.

You can create new definitions in a buffer, and edit and execute them until they are correct.

Related Commands

DEFINE PARAMETER
DELETE PARAMETER
SHOW PARAMETER

Example

```
LSE> EXTRACT PARAMETER id
```

Places the current definition of the *id* parameter at the end of the current buffer.

EXTRACT PLACEHOLDER

Extracts the definition of the named placeholder and formats the definition as a command.

Format

EXTRACT PLACEHOLDER *placeholder-name*

Qualifiers
/LANGUAGE[=language-name]
/NEW

Qualifiers

/LANGUAGE[=language-name]
Specifies the language associated with the placeholder being extracted. If you do not specify the /LANGUAGE qualifier, the default is the current language.

/NEW
Specifies that only the placeholder definitions defined during this editing session should be extracted.

Parameter

placeholder-name
Specifies the name of the placeholder you want. You may specify a wildcard.

Description

The EXTRACT PLACEHOLDER command extracts the named placeholder definition and formats it as a command. LSE inserts the indicated definitions at the end of the current buffer in a form that permits them to be read back and to replace existing definitions. Specifically, the DELETE PLACEHOLDER command precedes the corresponding DEFINE PLACEHOLDER command.

With the EXTRACT PLACEHOLDER command, you can modify definitions by editing and then executing them using the DO command. You can write definitions to a file.

You can use this command to extract placeholders to make global changes to them, such as changing delimiters or placeholder names. Use the SET NOAUTO_ERASE command to avoid erasing the placeholders as you type within their delimiters and perform other edits. Once you have edited the buffer, use the DO command to execute the changes.

You can create new definitions in a buffer, and edit and execute them until they are correct.

Related Commands

DEFINE PLACEHOLDER
DELETE PLACEHOLDER
SHOW PLACEHOLDER

Example

```
LSE> EXTRACT PLACEHOLDER/LANGUAGE=ADA text
```

Extracts the current definition of the *text* placeholder from the list of placeholders associated with the Ada language and places the definition at the end of the current buffer.

EXTRACT ROUTINE

Extracts the definition of the named routine and formats the definition as a command.

Format

EXTRACT ROUTINE *routine-name*

Qualifiers
/LANGUAGE[=language-name]
/NEW

Qualifiers

/LANGUAGE[=language-name]
Specifies the language associated with the routine being extracted. If you do not specify the /LANGUAGE qualifier, the default is the current language.

/NEW
Specifies that only the definitions of routines defined during this editing session should be extracted.

Parameter

routine-name
Specifies the name of the routine you want. You may specify a wildcard.

Description

The EXTRACT ROUTINE command extracts the named routine definition and formats it as a command. LSE inserts the indicated definitions at the end of the current buffer in a form that permits them to be read back and to replace existing definitions. Specifically, the DELETE ROUTINE command precedes the corresponding DEFINE ROUTINE command.

With the EXTRACT ROUTINE command, you can modify definitions by editing and then executing them using the DO command. You can write definitions to a file.

You can create new definitions in a buffer, and edit and execute them until they are correct.

Related Commands

DEFINE ROUTINE
DELETE ROUTINE
SHOW ROUTINE

Example

```
LSE> EXTRACT ROUTINE add_holder
```

Places the current definition of the *add_holder* routine at the end of the current buffer.

EXTRACT TAG

Extracts the definition of the named tag and formats the definition as a command.

Format

EXTRACT TAG *tag-name*

Qualifiers
/LANGUAGE[=language-name]
/NEW

Qualifiers

/LANGUAGE[=language-name]
Specifies the language associated with the tag being extracted. If you do not specify the /LANGUAGE qualifier, the default is the current language.

/NEW
Specifies that only the tag definitions defined during this editing session should be extracted.

Parameter

tag-name
Specifies the name of the tag you want. You may specify a wildcard.

Description

The EXTRACT TAG command extracts the named tag definition and formats it as a command. LSE inserts the indicated definitions at the end of the current buffer in a form that permits them to be read back and to replace existing definitions. Specifically, the DELETE TAG command precedes the corresponding DEFINE TAG command.

With the EXTRACT TAG command, you can modify definitions by editing and then executing them using the DO command. You can write definitions to a file.

You can create new definitions in a buffer, and edit and execute them until they are correct.

Related Commands

DEFINE TAG
DELETE TAG
SHOW TAG

Example

```
LSE> EXTRACT TAG/LANGUAGE=ADA text
```

Extracts the current definition of the *text* tag from the list of tags associated with the Ada language and places the definition at the end of the current buffer.

EXTRACT TOKEN

Extracts the definition of the named token and formats the definition as a command.

Format

EXTRACT TOKEN *token-name*

Qualifiers
/LANGUAGE[=language-name]
/NEW

Qualifiers

/LANGUAGE[=language-name]
Specifies the language associated with the token being extracted. If you do not specify the /LANGUAGE qualifier, the default is the current language.

/NEW
Specifies that only the definitions of tokens defined during this editing session should be extracted.

Parameter

token-name
Specifies the name of the token you want. You may specify a wildcard.

Description

The EXTRACT TOKEN command extracts the named token definition and formats it as a command. LSE inserts the indicated definitions at the end of the current buffer in a form that permits them to be read back and to replace existing definitions. Specifically, the DELETE TOKEN command precedes the corresponding DEFINE TOKEN command.

With the EXTRACT TOKEN command, you can modify definitions by editing and then executing them using the DO command. You can write definitions to a file.

You can use this command to extract tokens to make global changes to them, such as changing delimiters or token names. Use the SET NOAUTO_ERASE command to avoid erasing the tokens as you type within their delimiters and perform other edits. Once you have edited the buffer, use the DO command to execute the changes.

You can create new definitions in a buffer, and edit and execute them until they are correct.

Related Commands

DEFINE TOKEN
DELETE TOKEN
SHOW TOKEN

Example

```
LSE>  EXTRACT TOKEN WHILE
```

Places the current definition of the *WHILE* statement at the end of the current buffer.

For additional examples, see the section about redefining language elements in the *VAX Language-Sensitive Editor and VAX Source Code Analyzer User Manual.*

FILL

Reformats the text within a selected range to put as much text on a line as possible. This command is particularly useful for comments and ordinary prose, but is not normally used with program code.

Format

FILL

Qualifiers	**Defaults**
/COMMENT_COLUMN=CONTEXT_DEPENDENT	/COMMENT_COLUMN= CONTEXT_DEPENDENT
/COMMENT_COLUMN=number	/COMMENT_COLUMN= CONTEXT_DEPENDENT

Qualifiers

/COMMENT_COLUMN=CONTEXT_DEPENDENT (D)
/COMMENT_COLUMN=number
Specifies that the comment column should be determined from the context. LSE uses the position of the commented segment in the first line of the selected range as the comment column.

Number specifies an explicit column number in which to align the comments. LSE aligns all commented segments in the selected range with this column; all paragraphs within the range have the same comment column setting. *Number* must be an integer in the range of from 1 to 131; also, the value must be consistent with the lengths of the comment delimiters used within the range.

For a text fill, LSE ignores this qualifier.

Description

The FILL command reformats the text in the selected range. The selected range is the text between the select marker (see the SET SELECT_MARK command) and the current cursor position. If you do not provide a selected range, the FILL command reformats the current paragraph. (Note that the current paragraph includes the text on all previous and subsequent lines until LSE encounters a completely blank line.) LSE preserves any blank lines you insert in the text.

If the buffer is associated with a language, and comment delimiters have been defined for the language, then LSE just reformats the commented segments of the lines in the selected range. If the buffer is not associated with a language, or there are no comment delimiters, then LSE performs a text fill.

The FILL command reformats a block of text so that as many complete words as possible fit on each line without exceeding the right margin. You can change the right margin with the SET RIGHT_MARGIN command. Except in comments, the FILL command indents the reformatted text to the LEFT_MARGIN setting.

When you issue the FILL command, LSE treats spaces, tabs, and carriage returns as word delimiters. LSE treats character sequences as whole words if it recognizes such sequences as placeholders.

Keypad Equivalent

Key	**Keypad Mode**
PF1-KP8 FILL	EDT LK201, EDT VT100, EVE LK201
None	EVE VT100

DECwindows Interface Equivalent

Pull-down menu: Format --> Fill

Related Commands

DEFINE LANGUAGE
SET SELECT_MARK
SET WRAP

Examples

The /COMMENT_COLUMN=CONTEXT_DEPENDENT qualifier (the default) is in effect in the following examples.

1.

```
IF (col >= R_Margin) THEN ! This is the start of an
    BEGIN                    ! extended end-of-line comment block
    i := i + 1 ;
    j := j + i ;   ! another comment
  !to be filled
```

Issuing the FILL command for this example of line comments produces the following format:

```
IF (col >= R_Margin) THEN ! This is the start of an extended
    BEGIN                 ! end-of-line comment block
    i := i + 1 ;
    j := j + i ;          ! another comment to be filled
```

Note that the first word after the start of the comment on the second line (the word *extended*) was used to fill out the first line.

2.

```
IF (col >= R_Margin) THEN (* This is the start of a *)
        BEGIN            (* bracketed comment sequence that *)
        VAR x: INTEGER;      (* extends over several lines   *)
```

Issuing the FILL command for this example of consecutive single line bracketed comments produces the following format:

```
IF (col >= R_Margin) THEN (* This is the start of a bracketed *)
        BEGIN             (* comment sequence that extends    *)
        VAR x: INTEGER;   (* over several lines               *)
```

FIND

Locates occurrences described by the current SCA libraries.

Format

FIND *query-expression*

Qualifiers	**Defaults**
/DESCRIPTION=string	
/[NO]DISPLAY[=(option, . . .)]	/DISPLAY=DEFAULT
/[NO]LOG	/LOG
/[NO]MODIFY[=query-name]	/NOMODIFY
/NAME=query-name	
/OUTPUT[=file-spec]	
/[NO]REPLACE	/NOREPLACE
/[NO]RESULT=option	/RESULT=DEFAULT
/[NO]SYNCHRONIZE	/NOSYNCHRONIZE

Qualifiers

/DESCRIPTION=string
Specifies a single line of text that is displayed along with the query name when the query is displayed by issuing the SHOW QUERY command.

/DISPLAY[=(option, . . .)]
/DISPLAY=DEFAULT (D)
/NODISPLAY
Indicates how much information SCA displays concerning query results. Use one or more of the following keywords to request specific information:

NAME	Symbol name
CLASS	Class of item
LINE_NUMBER	Compilation line number

MODULE	Module name containing a symbol occurrence
FILE_SPEC	File name and type containing a symbol occurrence
FULL_FILE_SPEC	Complete file specification containing a symbol occurrence
RECORD_NUMBER	Record number within a source file
RELATIONSHIP	Relationship type
NUMBER	Number of the display line
OCCURRENCE_TYPE	Type of symbol occurrence (such as declaration, read, call)
ALL	All of the previous options
DEFAULT	Default settings of the display options
NONE	Nothing (equivalent to the /NODISPLAY qualifier)

You can prefix any keyword (except ALL, DEFAULT, and NONE) with NO to request that the information be excluded.

The initial default for each type of new query is as follows:

```
DISPLAY=(NAME,CLASS,MODULE,LINE,OCCURRENCE,RELATIONSHIP)
```

/LOG (D)
/NOLOG
Indicates whether the count of symbol occurrences will be reported.

/MODIFY[=query-name]
/NOMODIFY (D)
Indicates that an existing query is to be modified. By default, each FIND command creates a new query.

The /MODIFY=query-name qualifier indicates that the specified query should be modified according to the specification of the FIND command. The specified query must already exist.

By default, the /MODIFY qualifier specifies the current query.

/NAME[=query-name]
Specifies the name of the query. If a query with the same name already exists, you must also specify the /REPLACE qualifier. If a query-name is not specified, then SCA assigns a unique name to the query. The query name can be a quoted string.

/OUTPUT[=file-spec]
Specifies that command output is to go to a file rather than be displayed on your screen (or go to a batch log file). The default output file specification is SCA.LIS.

/REPLACE
/NOREPLACE (D)
Indicates whether existing queries should be replaced by new queries. By default, a FIND command that creates a query with the same name as an already existing query will fail.

/RESULT=option
/RESULT=DEFAULT (D)
/NORESULT
Indicates the type of query results displayed. You must specify one of the following keywords:

SYMBOLS	Only symbols are displayed.
OCCURRENCES	Symbols and occurrences are displayed.
DEFAULT	Either symbols or occurrences, or both are displayed. SCA chooses the result type that is most appropriate for the current query.

The /NORESULT qualifier specifies that no results should be displayed. This means that no query evaluation is done. If a query result exists because you issued a FIND command, then specifying /NORESULT causes that result to be deleted.

/SYNCHRONIZE
/NOSYNCHRONIZE (D)
Indicates that the query result must be synchronized with the current state of the virtual library being queried. By default, /NOSYNCHRONIZE causes SCA to do as little processing as necessary to evaluate the query. This can lead to query results that reflect the state of the virtual library at the time of a previous query.

The /SYNCHRONIZE qualifier specifies that the query result must be synchronized with the current virtual library. SCA attempts to minimize the amount of processing, but the result is still synchronized with the virtual library that was in effect at the time the query was evaluated.

FIND
SCA Command

Parameter

query-expression
Specifies the set of occurrences to be found.

For information on query expressions, see the chapters on query expressions and query language in the *VAX Language-Sensitive Editor and VAX Source Code Analyzer User Manual.*

Description

The FIND command locates occurrences described by the current SCA libraries. By default, each time you issue a FIND command, SCA creates a new query to describe the result. To remove queries you no longer need, use the DELETE QUERY command.

For more information about the FIND command, see the chapter on performing SCA tasks in the *VAX Language-Sensitive Editor and VAX Source Code Analyzer User Manual.*

DECwindows Interface Equivalent

FIND SYMBOL
Pull-down menu: Navigate --> Find Symbol

Related Commands

COLLAPSE
DELETE QUERY
EXPAND
GOTO QUERY
GOTO SOURCE
NEXT QUERY
NEXT STEP
PREVIOUS QUERY
PREVIOUS STEP

EXAMPLES

1.
```
LSE> FIND build*
```
Finds all occurrences of symbols whose name begins with *build*.

2.
```
LSE> FIND/RESULT=SYMBOL copy_file and symbol=literal
```
Finds all occurrences of literals named *copy_file*. Only symbol information is included in the display.

3.
```
LSE> FIND/RESULT=OCCURRENCE occ=primary and symbol=routine
```
Finds the primary declarations of all routines. Both symbol and occurrence information are included in the display.

4.
```
LSE> FIND calling expand_string
```
Finds the routines that are calling *expand_string*.

5.
```
LSE> FIND called_by( translit, depth=all )
```
Displays the complete call-tree below *translit*.

6.
```
LSE> FIND typed_by( integer, symbol=variable )
```
Finds all the variables of type *integer*.

FOCUS

Displays an overview of the buffer. The current line remains visible, and the rest of the buffer is compressed.

Format

FOCUS

Description

The FOCUS command displays the current line and its surrounding text. The rest of the lines in the buffer are collapsed as much as possible and represented by overview lines.

The editor determines the relative level of detail of a line by comparing the indentation of the line with the indentation of other lines. The editor's treatment of the indentation of a line is influenced by indentation adjustment definitions. For more information, see the DEFINE ADJUSTMENT command.

Keypad Equivalent

Key	**Keypad Mode**
PF1-period	All

DECwindows Interface Equivalent

Pull-down menu: View --> Focus

Related Commands

COLLAPSE
DEFINE ADJUSTMENT
DEFINE LANGUAGE/OVERVIEW_OPTIONS
EXPAND
MODIFY LANGUAGE
SET NOOVERVIEW
SET OVERVIEW
VIEW SOURCE

GOTO BOTTOM

Moves the cursor to the bottom of the current buffer.

Format

GOTO BOTTOM

Description

The GOTO BOTTOM command moves the cursor to the bottom of the current buffer. To achieve the same result, DECwindows interface users can use MB1 to drag the vertical scroll bar slider to the bottom of the scroll bar.

Keypad Equivalent

Key	**Keypad Mode**
PF1-KP4 BOTTOM	EDT LK201, EDT VT100, EVE LK201
PF1-E6	EDT LK201
PF1-↓	EVE LK201, EVE VT100

Related Commands

GOTO TOP

GOTO BUFFER

Moves the cursor to the specified buffer.

Format

GOTO BUFFER *buffer-name*

Qualifiers	**Defaults**
/[NO]CREATE	/NOCREATE
/[NO]READ_ONLY	/READ_ONLY
/[NO]WRITE	/NOWRITE

Qualifiers

/CREATE
/NOCREATE (D)
Specifies whether or not the buffer should be created if it does not exist.

/READ_ONLY (D)
/NOREAD_ONLY
Specifies whether or not the indicated buffer should have the read-only attribute. If the buffer has this attribute, then LSE does not write the contents to a file when you exit from LSE or when you issue a COMPILE command. This qualifier has an effect only if the GOTO BUFFER command is creating a buffer. If you are going to an already existing buffer, the read-write status of that buffer is not changed. The /WRITE qualifier is equivalent to the /NOREAD_ONLY qualifier.

/WRITE
/NOWRITE (D)
Specifies whether or not the indicated buffer should have the write attribute. If the buffer has this attribute, then LSE writes the contents of the buffer to a file when you exit from LSE or when you issue a COMPILE command. This qualifier has an effect only if the GOTO BUFFER command is creating a buffer. If you are going to an already existing buffer, the read-write status

of the buffer is not changed. The /NOREAD_ONLY qualifier is equivalent to the /WRITE qualifier.

Parameter

buffer-name
Specifies the name of the buffer. You may use abbreviations.

You can specify a buffer name with a character string value of up to 255 alphanumeric or special characters. If you begin the buffer name with special characters, such as those accessed on the top row of your keyboard by pressing the shift key, you must enclose the buffer name in quotation marks. Similarly, to specify a name that contains embedded blanks (spaces), or quotation marks and spaces, enclose the entire string in quotation marks.

Description

The GOTO BUFFER command moves the cursor to the specified buffer. LSE maps the buffer to the current window, and moves the cursor to the last remembered position in that buffer.

You can use the mouse to select a buffer from the list displayed by the SHOW BUFFERS command.

Related Commands

GOTO FILE
NEXT BUFFER
PREVIOUS BUFFER
SHOW BUFFERS

Example

```
LSE>  GOTO BUFFER $SHOW
```

Causes LSE to display the buffer that contains the latest response to a SHOW command.

GOTO CHARACTER

Moves the cursor to the next character.

Format

GOTO CHARACTER

Qualifiers	**Defaults**
/CURRENT	/CURRENT
/FORWARD	/CURRENT
/HORIZONTALLY	/HORIZONTALLY
/REVERSE	/CURRENT
/VERTICALLY	/HORIZONTALLY

Qualifiers

/CURRENT (D)
Instructs LSE to use the current direction of the buffer.

/FORWARD
Instructs LSE to move the cursor down or to the right.

/HORIZONTALLY (D)
Instructs LSE to move the cursor horizontally.

/REVERSE
Instructs LSE to move the cursor up or to the left.

/VERTICALLY
Instructs LSE to move the cursor vertically.

GOTO CHARACTER

Description

The GOTO CHARACTER command moves the cursor one character in the indicated direction. LSE does not position the cursor when the screen is empty, unless text spaces have been created using the space bar. The cursor moves across tab characters and wraps at the edge of the screen.

You can use the mouse cursor to position the editing cursor to any text in an editing window.

Keypad Equivalent

GOTO CHARACTER/VERTICALLY/FORWARD

Key	Keypad Mode
Down ⇩	All
KP2 ⇩	EVE VT100

GOTO CHARACTER/HORIZONTALLY/REVERSE

Key	Keypad Mode
Left ⇦	All
KP1 ⇦	EVE VT100

GOTO CHARACTER/HORIZONTALLY/FORWARD

Key	Keypad Mode
Right ⇨	All
KP3 ⇨	EVE VT100

GOTO CHARACTER/VERTICALLY/REVERSE

Key	Keypad Mode
Up $\boxed{\uparrow}$	All
KP5 $\boxed{\uparrow}$	EVE VT100

GOTO CHARACTER/HORIZONTALLY/CURRENT

Key	Keypad Mode
KP3 $\boxed{\text{CHAR}}$	EDT LK201, EDT VT100, EVE LK201
None	EVE VT100

Related Commands

GOTO LINE
GOTO WORD

GOTO COMMAND

Produces the LSE Command> prompt at which you can enter LSE or SCA commands.

Format

GOTO COMMAND

Description

The GOTO COMMAND command moves the cursor to the command region. With the DECwindows interface, you can use the mouse to move the cursor to the commands region.

Keypad Equivalent

Key	**Keypad Mode**
Do DO	EDT LK201, EVE LK201
PF1-KP7	All

Related Commands

DO

GOTO DECLARATION

Displays the declaration of the symbol indicated. LSE displays the source code containing the symbol declaration in another window and positions the cursor on the symbol declaration.

Format

GOTO DECLARATION *[symbol-name]*

Qualifiers	**Defaults**
/ASSOCIATED	/PRIMARY
/CONTEXT_DEPENDENT	/PRIMARY
/INDICATED	
/PRIMARY	/PRIMARY

Qualifiers

/ASSOCIATED
Indicates that you want to see the associated declaration for the symbol. An associated declaration is a related declaration that accompanies the primary declaration (such as an EXTERNAL declaration).

/CONTEXT_DEPENDENT
If you specify both the /CONTEXT_DEPENDENT and the /INDICATED qualifiers, then SCA determines which declaration to display by using the following criteria:

- If the indicated occurrence of the symbol is a reference, LSE displays the declaration specified by the compiler as bound to that occurrence of the symbol.
- If the indicated occurrence of the symbol is an associated declaration, LSE displays the primary declaration.
- If the indicated occurrence of the symbol is a primary declaration, LSE displays the associated declaration.

If you specify the /CONTEXT_DEPENDENT qualifier but not the /INDICATED qualifier, then LSE displays the primary declaration.

/INDICATED
Instructs LSE to use the symbol name at the current cursor position, or the text within the currently active selected range, as the symbol name. To help SCA identify exactly which occurrence of the symbol name the cursor is positioned on, LSE passes both the current cursor position in the buffer and the file specification for the current buffer to SCA.

If SCA has no information for the symbol name at the current cursor position (for example, if the line containing the symbol is a new line and the file has not been recompiled), then SCA uses whatever general information it has about that symbol as if you issued a GOTO DECLARATION command for the symbol name without the /INDICATED qualifier.

If you specify the /INDICATED qualifier, you must not specify the *symbol-name* parameter.

/PRIMARY (D)
Indicates that you want to see the primary declaration for the symbol. A primary declaration is the declaration that SCA interprets as most significant for a symbol (such as a FUNCTION declaration). For example, the primary declaration of a routine describes the body of the routine.

Parameter

symbol-name
Specifies that the declaration associated with the named symbol is to be displayed. You must not specify a symbol name if you specify the /INDICATED qualifier.

Description

The GOTO DECLARATION command causes LSE to display the source for the declaration of the specified or indicated symbol.

If more than one declaration is to be displayed, LSE creates a new query to list those declarations.

Keypad Equivalent

GOTO DECLARATION/INDICATED/PRIMARY

Key	Keypad Mode
CTRL/D	All

GOTO DECLARATION/INDICATED/CONTEXT_DEPENDENT

Key	Keypad Mode
PF1-CTRL/D	All

DECwindows Interface Equivalent

GOTO DECLARATION/INDICATED
Pop-up menu: User buffer --> Find Declaration
Pull-down menu: Navigate --> Find Declaration

Related Commands

FIND
GOTO QUERY
GOTO SOURCE

Example

```
LOCAL   X;
    .
    .
    .
X = Y;
```

```
LSE> GOTO DECLARATION/INDICATED
```

Causes LSE to display the declaration *LOCAL X* if your cursor is positioned on the X of the assignment statement X = Y.

GOTO FILE

Moves the cursor to the buffer containing the specified file. If no buffer contains the specified file, LSE reads the file into a new buffer.

Format

GOTO FILE *file-spec*

Qualifiers	**Default**
/[NO]CREATE	/NOCREATE
/[NO]MODIFY	
/NEW	
/READ_ONLY	
/WRITE	

Qualifiers

/CREATE
/NOCREATE (D)
Specifies whether the GOTO FILE command should succeed if the specified file does not exist. This qualifier has no effect if you are going to an existing buffer.

/MODIFY
/NOMODIFY
Specifies whether the buffer you create is modifiable or unmodifiable. If you specify the /MODIFY qualifier, the GOTO FILE command creates a modifiable buffer. If you specify the /NOMODIFY qualifier, the GOTO FILE command creates an unmodifiable buffer. If you do not specify either qualifier, LSE determines the buffer's modifiable status from the read-only/write setting. By default, a read-only buffer is unmodifiable and a write buffer is modifiable.

/NEW
Specifies that you want to create a new file. If the specified file already exists, LSE reports an error and aborts the command. The *file-spec* parameter may not contain wildcards if you specify this qualifier. You cannot use this qualifier with the /[NO]CREATE or /[NO]MODIFY qualifiers.

/READ_ONLY
Specifies that the buffer you create is read-only and therefore unmodifiable. This qualifier and the /WRITE qualifier override any setting established by the SET DIRECTORY command.

If you specify neither the /READ_ONLY nor the /WRITE qualifier, LSE uses the default established by the most recent SET DIRECTORY command for the directory that contains the file. If during your current editing session you have not issued a SET DIRECTORY command nor defined the logical LSE$READ_ONLY_DIRECTORY, then the buffer is writeable by default.

/WRITE
Specifies that the buffer you create is writeable and therefore modifiable. This qualifier and the /READ_ONLY qualifier override any setting established by the SET DIRECTORY command.

If you specify neither the /WRITE nor the /READ_ONLY qualifier, LSE uses the default established by the most recent SET DIRECTORY command for the directory that contains the file. If during your current editing session you have not issued a SET DIRECTORY command nor defined the logical LSE$READ_ONLY_DIRECTORY, then the buffer is writeable by default.

Parameter

file-spec
Specifies the name of the file to be edited. LSE uses the directories specified in the SET SOURCE_DIRECTORY command to resolve the file specification. If the file cannot be found in one of those directories (or the list of directories is empty) and you used the /CREATE qualifier, LSE creates the file in your default directory.

Description

The GOTO FILE command moves the cursor to its last position in the buffer containing the specified file if a buffer corresponding to the named file already exists.

If no such buffer exists, LSE creates a new one, taking the buffer name from the name and type of the *file-spec* parameter. If that name is not unique, LSE prompts you for a buffer name and gives you the option of replacing an already existing buffer of the same name or canceling the command. If you do not cancel the command, LSE then reads the indicated file into the buffer, positions the cursor in that buffer, and maps the buffer to the current window.

If you do not specify either the /READ_ONLY or the /WRITE qualifier on the command, LSE sets the read/write status of the buffer based on the status of the directory in which the file is found. If the directory is a read-only directory (that is, if it is on the list established by the SET DIRECTORY/READ_ONLY command), then LSE creates the buffer as read-only and unmodifiable; otherwise, the buffer is set writeable and modifiable.

If the indicated file is to be read in (that is, it is not already in a buffer), LSE uses CMS to fetch a copy of the file and place it in an unmodifiable buffer if the directory for the file to be accessed is the same as your current CMS library. The GOTO FILE command uses the setting of the SET CMS command when performing a FETCH operation.

Note that you cannot use the GOTO FILE command to reserve files from your current CMS library. To reserve a file, use the RESERVE command.

DECwindows Interface Equivalent

GOTO FILE/NEW
Pull-down menu: File --> New . . .

GOTO FILE
Pull-down menu: File --> Open . . .

Related Commands

GOTO BUFFER
READ
SET CMS
SET DIRECTORY

Example

```
LSE> GOTO FILE x.y
```

Brings the file *x.y* into the current buffer.

GOTO LINE

Moves the cursor to the end of the line, or the next line if the cursor is already at the end of a line.

Format

GOTO LINE

Qualifiers	Defaults
/BEGINNING	/BEGINNING
/BOUND	
/BREAK	
/CURRENT	/CURRENT
/END	/BEGINNING
/FORWARD	/CURRENT
/REVERSE	/CURRENT

Qualifiers

/BEGINNING (D)
Indicates that the cursor should be moved to the beginning of the line. The /BEGINNING, /BREAK, /BOUND, and /END qualifiers are mutually exclusive.

/BOUND
Moves the cursor to the beginning or the end of the current line depending on whether the direction specified is FORWARD or REVERSE. If the cursor is already at the specified destination, LSE issues a message to that effect and the cursor does not move. The /BEGINNING, /BREAK, /BOUND, and /END qualifiers are mutually exclusive.

/BREAK
Moves the cursor either to the beginning or end of a line depending on whether the direction currently specified is FORWARD or REVERSE. If the cursor is already at the specified destination, LSE moves it to

the corresponding break on the next line in the current direction. The /BEGINNING, /BREAK, /BOUND, and /END qualifiers are mutually exclusive.

/CURRENT (D)
Instructs LSE to use the current direction of the buffer.

/END
Indicates that the cursor should be moved to the end of the line. The /BEGINNING, /BREAK, /BOUND, and /END qualifiers are mutually exclusive.

/FORWARD
Instructs LSE to move the cursor down or to the right.

/REVERSE
Instructs LSE to move the cursor up or to the left.

Description

The GOTO LINE command moves the cursor to one end of the line in the direction indicated. If the cursor is already at the end of the current line, this command moves the cursor to the next line, unless you have specified the /BOUND qualifier.

Keypad Equivalent

GOTO LINE/BEGINNING/REVERSE

Key	Keypad Mode
CTRL/H BACKSPACE	EDT LK201, EDT VT100
F12 BOL	EDT LK201

GOTO LINE

GOTO LINE/BEGINNING/CURRENT

Key	Keypad Mode
KP0 LINE	EDT LK201, EDT VT100, EVE LK201
None	EVE VT100

GOTO LINE/END/CURRENT

Key	Keypad Mode
KP2 EOL	EDT LK201, EDT VT100, EVE LK201
None	EVE VT100

GOTO LINE/BOUND/REVERSE

Key	Keypad Mode
CTRL/H BACKSPACE	EVE LK201, EVE VT100
PF1-←	EVE VT100, EVE LK201

GOTO LINE/BOUND/FORWARD

Key	Keypad Mode
CTRL/E	EVE LK201, EVE VT100
PF1-→	EVE VT100, EVE LK201

GOTO LINE/BREAK/CURRENT

Key	Keypad Mode
F12 MOVE BY LINE	EVE LK201
Keypad minus (−) MOVE BY LINE	EVE VT100

Related Commands

GOTO CHARACTER
GOTO WORD

Example

```
LSE>  GOTO LINE/BOUND/REVERSE
```

Moves the cursor to the start of the current line. If the cursor is at the start of a line, LSE displays the message “Already at the start of the line” when you enter this command.

GOTO MARK

Moves the cursor to a marker name defined by a SET MARK command.

Format

GOTO MARK *marker-name*

Parameter

marker-name
Specifies the name of a marker created with a SET MARK command.

Description

The GOTO MARK command moves the cursor to a marker name you define using a SET MARK command. LSE maps a new buffer to the current window if the marker you specify is not in the current buffer.

Related Commands

SET MARK

Example

```
LSE> GOTO MARK 1
```

Moves the cursor to the position previously marked using the command SET MARK 1. If MARK *1* is not in the current buffer, the buffer that contains MARK *1* becomes the current buffer.

GOTO PAGE

Moves the cursor to the next page where a page boundary is a form feed or the beginning or end of a buffer.

Format

GOTO PAGE

Qualifiers	Defaults
/CURRENT	/CURRENT
/FORWARD	/CURRENT
/REVERSE	/CURRENT

Qualifiers

/CURRENT (D)
Instructs LSE to use the current direction of the buffer.

/FORWARD
Instructs LSE to move the cursor down.

/REVERSE
Instructs LSE to move the cursor up.

Description

The GOTO PAGE command moves the cursor to the beginning of the next or previous page in the current buffer, depending on the direction set by FORWARD or REVERSE. A form feed delimits a page. If there is no form feed in the current buffer, the GOTO PAGE command moves the cursor to the end (or beginning) of the buffer.

GOTO PAGE

Keypad Equivalent

Key	Keypad Mode
KP7 PAGE	EDT LK201, EDT VT100, EVE LK201
None	EVE VT100

Related Commands

GOTO LINE
GOTO WORD

GOTO PLACEHOLDER

Moves the cursor to a placeholder.

Format

GOTO PLACEHOLDER

Qualifiers	Defaults
/ALL	/ALL
/CURRENT	/CURRENT
/FORWARD	/CURRENT
/NOPSEUDOCODE	
/REVERSE	/CURRENT

Qualifiers

/ALL (D)
Instructs the GOTO PLACEHOLDER command to recognize all placeholders, including pseudocode placeholders and overview records.

/CURRENT (D)
Instructs LSE to use the current direction of the buffer.

/FORWARD
Instructs LSE to move the cursor down or to the right.

/NOPSEUDOCODE
Instructs the GOTO PLACEHOLDER command to ignore pseudocode placeholders.

/REVERSE
Instructs LSE to move the cursor up or to the left.

GOTO PLACEHOLDER

Description

The GOTO PLACEHOLDER command moves the cursor to the next placeholder in the direction indicated. A placeholder must be defined for the GOTO PLACEHOLDER command to recognize it.

Keypad Equivalent

GOTO PLACEHOLDER/ALL/FORWARD

Key	**Keypad Mode**
CTRL/N	All

GOTO PLACEHOLDER/ALL/REVERSE

Key	**Keypad Mode**
CTRL/P	All

GOTO PLACEHOLDER/NOPSEUDOCODE/FORWARD

Key	**Keypad Mode**
PF1-CTRL/N	All

GOTO PLACEHOLDER/NOPSEUDOCODE/REVERSE

Key	**Keypad Mode**
PF1-CTRL/P	All

Related Commands

DEFINE PLACEHOLDER
ERASE PLACEHOLDER

GOTO QUERY

Moves the cursor to the specified SCA query session.

Format

GOTO QUERY *query-name*

Parameter

query-name
Specifies the name of the query session.

Description

The GOTO QUERY command splits the current window (if possible) and maps the named query to the current window and the buffer associated with the query to the screen.

Related Commands

DELETE QUERY
FIND
NEXT QUERY
PREVIOUS QUERY
SHOW QUERY

Example

```
LSE> GOTO QUERY 1
```

Moves the cursor to the window containing query buffer *1*.

GOTO REVIEW

Moves the cursor to the currently active review session.

Format

GOTO REVIEW

Description

The GOTO REVIEW command moves the cursor to the current review session and sets the current status to review mode. LSE maps the $REVIEW buffer to the screen and positions the cursor to the last current position in that buffer.

If no review session is currently active, the GOTO REVIEW command fails.

Related Commands

END REVIEW
GOTO QUERY
GOTO SOURCE
NEXT STEP
PREVIOUS STEP
REVIEW

GOTO SCREEN

Moves the cursor in the indicated direction two lines less than the number of lines in the current window.

Format

GOTO SCREEN

Qualifiers	Defaults
/CURRENT	/CURRENT
/FORWARD	/CURRENT
/REVERSE	/CURRENT

Qualifiers

/CURRENT (D)
Instructs LSE to use the current direction of the buffer.

/FORWARD
Instructs LSE to move the cursor down.

/REVERSE
Instructs LSE to move the cursor up.

Description

The GOTO SCREEN command moves the cursor two lines less than the number of lines in the current window, depending on the direction set by FORWARD or REVERSE.

Users of the DECwindows interface can achieve similar results by pressing MB1 above or below the slider in the vertical scroll bar.

GOTO SCREEN

Keypad Equivalent

GOTO SCREEN/FORWARD

Key	Keypad Mode
E6 NEXT SCREEN	EDT LK201, EVE LK201
KP0 Next Screen	EVE VT100
None	EDT VT100

GOTO SCREEN/REVERSE

Key	Keypad Mode
E5 PREV SCREEN	EDT LK201, EVE LK201
Keypad period Prev Screen	EVE VT100
None	EDT VT100

GOTO SCREEN/CURRENT

Key	Keypad Mode
KP8 SECT	EDT LK201, EDT VT100, EVE LK201
None	EVE VT100

GOTO SOURCE

Displays the source corresponding to the current diagnostic or query item. To display a query item, you must be using SCA.

Format

GOTO SOURCE

Qualifiers
/READ_ONLY
/WRITE

Qualifiers

/READ_ONLY
Specifies that the buffer containing the source be set read-only and therefore unmodifiable. Using this qualifier overrides any setting established by the SET DIRECTORY command.

/WRITE
Specifies that the buffer containing the source be set writeable and therefore modifiable. Using this qualifier overrides any setting established by the SET DIRECTORY command.

Description

The GOTO SOURCE command has different actions depending on whether LSE is in review or query mode. To be in query mode, you must be using SCA.

Review Mode

In review mode, LSE selects the diagnostic at the current position in the buffer $REVIEW and a region where you want the source displayed. This becomes the current diagnostic.

LSE highlights the current diagnostic and the current region and displays in a second window, with the region highlighted, the file containing the current region. When a diagnostic is selected in this way, the buffer containing the current region becomes the current buffer.

LSE may display a suggested error correction and prompt for a yes (Y) or no (N) response; LSE makes the correction if you respond with a Y.

Query Mode

In query mode, LSE selects the query item occurrence at the current position in the current query buffer. This becomes the current query item. LSE highlights the current query item and displays the file containing the corresponding source for the current query item in a second window. The buffer containing the source that corresponds to the current query item becomes the current buffer.

Review or Query Modes

If the source file corresponding to the current diagnostic region or current query item is not in a buffer, LSE creates an unmodifiable buffer and reads the source file specified in the diagnostics file or SCA data file into that buffer.

If it cannot find that file, LSE uses the list of directories specified by the SET SOURCE_DIRECTORY command to find the file.

LSE uses CMS to access a file if the directory for the file to be accessed is the same as the translation of CMS$LIB.

Users of the DECwindows interface can invoke the GOTO SOURCE command by moving the mouse cursor to an occurrence in the query buffer, or an error region in the review buffer, and pressing MB1 twice.

Keypad Equivalent

Key	Keypad Mode
CTRL/G	All

DECwindows Interface Equivalent

Pop-up menu: $\left\{ \begin{array}{l} \text{Review buffer --> Goto Source} \\ \text{Query buffer --> Goto Source} \end{array} \right\}$

Double click MB1 on the review or query item.

Related Commands

SET DIRECTORY
SET SOURCE_DIRECTORY
SHOW DIRECTORY
SHOW SOURCE_DIRECTORY

Example

```
LSE>  GOTO SOURCE
```

Moves the cursor to the buffer containing the source code corresponding to the current diagnostic or query item.

GOTO TOP

Moves the cursor to the top of the current buffer.

Format

GOTO TOP

Description

The GOTO TOP command moves the cursor to the top of the buffer that contains the cursor. To achieve the same result, DECwindows interface users can use MB1 to drag the vertical scroll bar slider to the top of the scroll bar.

Keypad Equivalent

Key	Keypad Mode
PF1-KP5 $\boxed{\text{TOP}}$	EDT LK201, EDT VT100, EVE LK201
PF1-E5	EDT LK201
PF1-↑	EVE LK201, EVE VT100

Related Commands

GOTO BOTTOM

GOTO WORD

Moves the cursor to the beginning of the current, next, or previous word in the current buffer, depending on the direction specified.

Format

GOTO WORD

Qualifiers	Defaults
/CURRENT	/CURRENT
/FORWARD	/CURRENT
/REVERSE	/CURRENT

Qualifiers

/CURRENT (D)
Instructs LSE to use the current direction of the buffer.

/FORWARD
Instructs LSE to move the cursor down or to the right.

/REVERSE
Instructs LSE to move the cursor up or to the left.

Description

The GOTO WORD command moves the cursor to the first character of the current, next, or previous word, depending on the current direction or the direction set by the /FORWARD or /REVERSE qualifiers. If the current direction is FORWARD, the cursor moves to the beginning of the next word. If the current direction is REVERSE, the cursor moves to the beginning of the current word; if the cursor is at the beginning of a word, it moves to the beginning of the previous word.

GOTO WORD

A word consists only of identifier characters and trailing blanks and can be delimited only by tabs or characters not specified in the /IDENTIFIER_CHARACTERS qualifier on the DEFINE LANGUAGE command. LSE also considers all nonblank, nonidentifier characters to be words.

Keypad Equivalent

Key	Keypad Mode
KP1 WORD	EDT LK201, EDT VT100, EVE LK201
None	EVE VT100

Related Commands

GOTO CHARACTER
GOTO LINE

HELP

Displays information about LSE and SCA commands.

Format

HELP *[topic-list]*

Qualifiers
/INDICATED
/KEYPAD
/LANGUAGE=language-name
/LIBRARY=library-name
/PACKAGE=package-name

Qualifiers

/INDICATED
Causes LSE to display the help text associated with the token, placeholder, or routine at the current cursor position. If you do not specify or negate the /LANGUAGE qualifier or the /PACKAGE qualifier, LSE first looks for a language element. If the indicated item is not a language element, then LSE looks for a package element.

The help text comes from the HELP library associated with the specified language or package. LSE forms a topic string by concatenating the /TOPIC_STRING qualifier associated with the language or package followed by the indicated token, placeholder, or entry name. LSE then searches for the topic in the HELP library.

You cannot use the /INDICATED qualifier with any of the following qualifiers: /KEYPAD, /LANGUAGE, /LIBRARY, or /PACKAGE.

/KEYPAD
Specifies that you want keypad HELP. You cannot use the /KEYPAD qualifier with any of the following qualifiers: /INDICATED, /LANGUAGE, /LIBRARY, or /PACKAGE.

The /KEYPAD qualifier builds the keypad diagram by using legends specified with the /LEGEND qualifier on the DEFINE KEY command. When the diagram is displayed and you press a key, LSE looks up the topic specified for that key by using the /TOPIC qualifier on the DEFINE KEY command, and displays the corresponding help text. The HELP library accessed is LSE$KEYPAD.HLB.

/LANGUAGE=language-name
Causes LSE to take the value of the /TOPIC_STRING qualifier for the indicated language and concatenate that value to the front of the *topic-list* parameter on the HELP command. If you specify the /LANGUAGE qualifier without a value, then LSE uses the language associated with the current buffer. (In this case, not having the current buffer associated with a language creates an error.)

You must not specify either the /KEYPAD qualifier or the /PACKAGE qualifier with the /LANGUAGE qualifier.

/LIBRARY=library-name
Specifies which HELP library LSE searches for the topic. This qualifier overrides the library file determined by LSE's default behavior. You may specify any other qualifiers with the /LIBRARY qualifier, except for the /KEYPAD qualifier.

/PACKAGE=package-name
Causes LSE to take the value of the /TOPIC_STRING qualifier for the indicated package and concatenate that value to the front of the *topic-list* parameter on the HELP command. You must provide the package name as the value of the qualifier.

You must not specify either the /KEYPAD or the /LANGUAGE qualifier with the /PACKAGE qualifier.

Parameter

topic-list
Indicates the topic for which you want help. This may be any list of topics valid for input to the DCL command interpreter's HELP command. The topic list must not be specified with the /INDICATED qualifier.

Description

The HELP command displays information about the requested topic of LSE, a language, or a package.

If you have more than one screen of help text available, and do not want to review the additional screens of information, press CTRL/Z to return to editing mode.

After exiting from HELP, the buffer $HELP contains the text displayed by the HELP command. This does not happen if you are using keypad HELP.

Keypad Equivalent

HELP/KEYPAD (VT100 keypad)

Key	Keypad Mode
PF2 HELP	All

HELP/KEYPAD (VT200 keypad)

Key	Keypad Mode
Help	EDT LK201, EVE LK201

HELP/INDICATED

Key	Keypad Mode
PF1-PF2 HELP IND	All
PF1-Help	EDT LK201, EVE LK201

Examples

1. `LSE> HELP CREATE LIBRARY`

   Invokes HELP at the LSE level.

2.
```
LSE> HELP/LANGUAGE=PASCAL STATEMENTS
```

Indicates that, for Pascal, the value *PASCAL* is assigned to the /TOPIC_STRING qualifier. LSE HELP is invoked to provide information about the *STATEMENTS* topic list.

INCLUDE

Inserts the specified file at the current editing position.

Format

INCLUDE *file-spec*

Qualifier
/BUFFER=buffer-name

Qualifier

/BUFFER=buffer-name
Specifies a buffer into which the file is to be included. If the buffer does not exist, it is created for display only (the buffer cannot be written back to a file).

Parameter

file-spec
Specifies the file to be copied to the current editing position. Wildcards are permitted in DECwindows mode.

Description

The INCLUDE command inserts the contents of the specified file at the current editing position. After inserting the file, the editing cursor is positioned on the first character of the inserted text.

This command is similar to the READ command, except that the INCLUDE command inserts the file's contents into the receiving buffer at the position your cursor was on. The cursor is then positioned on the first character of inserted text rather than remaining on the original character.

INCLUDE

DECwindows Interface Equivalent

Pop-up menu: None
Pull-down menu: File --> Include . . .

Related Commands

READ

Example

```
LSE> INCLUDE y.x
```

Opens file *y.x* for input and inserts its contents at the current editing position, leaving the cursor on the first character of the inserted text.

INSPECT

Inspects the consistency between declarations or references for the same symbol.

Format

INSPECT *query-expression*

Qualifiers	**Defaults**
/CHARACTERISTICS=(option[. . .])	/CHARACTERISTICS=ALL
/DESCRIPTION=string	
/[NO]DISPLAY[=(option, . . .)]	/DISPLAY=DEFAULT
/[NO]ERROR_LIMIT=(global-limit[,symbol-limit])	/NOERROR_LIMIT
/[NO]LOG	/LOG
/[NO]MODIFY[=query-name]	/NOMODIFY
/NAME=query-name	
/OUTPUT[=file-spec]	
/[NO]REPLACE	/NOREPLACE
/[NO]RESULT=option	/RESULT=DEFAULT
/SEVERITY_LEVEL=severity-level	/SEVERITY=INFORMATIONAL
/[NO]SYNCHRONIZE	/NOSYNCHRONIZE

Qualifiers

/CHARACTERISTICS=(option[. . .])
/CHARACTERISTICS=ALL (D)
Indicates which characteristics of the occurrences should be checked. You can use one or more of the following options to request specific information:

IMPLICIT_DECLARATIONS	Checks that all symbols are explicitly declared.
TYPE	Checks that the types of all occurrences of each symbol match.

UNIQUENESS	Checks that multiple declarations of the same symbol have the same name.
UNUSED_SYMBOLS	Checks that all symbols are used.
USAGE	Looks for symbols that are read but never written, or written but never read.
ALL	Checks all of the preceding characteristics.

Any of these options (except ALL) can have the prefix NO to indicate that the characteristic should not be checked.

Each of the characteristic options takes a *query-expression* as an optional value. The characteristic-specific query expression specifies the set of occurrences for which that characteristic will be checked. If the prefix NO is present, then the query expression indicates occurrences for which that characteristic will not be checked. The default query expression for each characteristic option is to check all occurrences.

/DESCRIPTION=string
Specifies a single line of text that is displayed along with the query name when the query is displayed by issuing the SHOW QUERY command.

/DISPLAY[=(option, . . .)]
/DISPLAY=DEFAULT (D)
/NODISPLAY
Indicates how much information SCA displays concerning query results. Use one or more of the following keywords to request specific information:

NAME	Symbol name
CLASS	Class of item
LINE_NUMBER	Compilation line number
FILE_NAME	File name and type containing a symbol occurrence
FULL_FILE_SPEC	Complete file specification containing a symbol occurrence
RECORD_NUMBER	Record number within a source file
OCCURRENCE_TYPE	Type of symbol occurrence (such as declaration, read, call)
ALL	All of the previous options
DEFAULT	Default settings of the display options
NONE	Nothing (equivalent to the /NODISPLAY qualifier)

You can prefix any keyword (except ALL, DEFAULT, and NONE) with NO to request that information be excluded.

The initial default for each type of new query is as follows:

```
DISPLAY=(NAME,CLASS,MODULE,LINE,OCCURRENCE)
```

/ERROR_LIMIT=(global-limit[,symbol-limit])
/NOERROR_LIMIT (D)

Specifies the maximum number of errors that the INSPECT command should report. This causes the INSPECT command to stop if the number of errors exceeds the maximum.

The *global-limit* parameter specifies the maximum number of errors reported for all symbols before the INSPECT command stops.

The *symbol-limit* parameter specifies the maximum number of errors reported for a particular symbol before the INSPECT command stops reporting errors for that symbol.

/LOG (D)
/NOLOG

Indicates whether the count of symbol occurrences will be reported.

/MODIFY[=query-name]
/NOMODIFY (D)

Indicates that an existing query is to be modified. By default, each INSPECT command creates a new query.

The /MODIFY=query-name qualifier indicates that the specified query should be modified according to the specification of the INSPECT command. The specified query must already exist.

By default, the /MODIFY qualifier specifies the current query.

/NAME[=query-name]

Specifies the name of the query. If a query with the same name already exists, you must also specify the /REPLACE qualifier. If a query name is not specified, then SCA assigns a unique name to the query.

/OUTPUT[=file-spec]

Specifies that command output is to go to a file rather than be displayed on your screen (or go to a batch log file). The default output file specification is SCA.LIS.

/REPLACE
/NOREPLACE (D)
Indicates whether existing queries should be replaced by new queries. By default, an INSPECT command that creates a query with the same name as an already existing query will fail.

/RESULT=option
/RESULT=DEFAULT (D)
/NORESULT
Indicates the type of query results displayed. You must specify one of the following keywords:

SYMBOLS	Only symbols are displayed.
OCCURRENCES	Symbols and occurrences are displayed.
DEFAULT	Either symbols or occurrences, or both are displayed. SCA chooses the result type that is most appropriate for the current query.

The /NORESULT qualifier specifies that no results should be displayed. This means that no query evaluation is done. If a query result exists because you issued an INSPECT command, then specifying /NORESULT causes that result to be deleted.

/SEVERITY_LEVEL=severity-level
/SEVERITY=INFORMATIONAL (D)
Indicates the lowest severity level for diagnostics to be reported, as follows:

INFORMATIONAL
WARNING
ERROR
FATAL_ERROR

/SYNCHRONIZE
/NOSYNCHRONIZE (D)
Indicates that the query result must be synchronized with the current state of the virtual library being queried. By default, /NOSYNCHRONIZE causes SCA to do as little processing as necessary to evaluate the query. This can lead to query results that reflect the state of the virtual library at the time of a previous query.

The /SYNCHRONIZE qualifier specifies that the query result must be synchronized with the current virtual library. SCA attempts to minimize the amount of processing, but the result is still synchronized with the virtual library that was in effect at the time the query was evaluated.

Parameter

query-expression
Specifies the set of occurences to be inspected.

Description

The INSPECT command checks the consistency between declarations or references for the same symbol.

Related Commands

FIND

Example

```
LSE>  INSPECT *
```

Inspects all characteristics of all symbols.

LINE

Moves the cursor in the current buffer to the start of the source line you specify.

Format

LINE *integer [procedure-name]*

Parameters

integer
Specifies the number of the line in the current buffer to which you want LSE to move the cursor. If you do not specify a line number, LSE prompts for one. Pressing CTRL/Z at the prompt cancels the command.

procedure-name
Specifies the name of a VAXTPU procedure in the current buffer. The procedure name is valid only for VAXTPU source files. This parameter is useful because some compiler messages refer to line numbers in a procedure.

To find out the current line number and total number of lines in the buffer, use the WHAT LINE command.

Description

The LINE command moves the cursor in the current buffer to the start of the line you specify. If the line requested is hidden, then the overview records are expanded to the source level and the cursor is placed on the requested line.

Related Commands

WHAT LINE

EXAMPLES

1.
```
LSE> LINE 14
```
Moves the cursor to the beginning of line 14.

2.
```
LSE> LINE 12 user_proc
```
Moves the cursor to the beginning of line 12 of a procedure named *USER_PROC*.

LOAD

Loads one or more files of compiler-generated source analysis data into an SCA library.

Format

LOAD *file-spec[, . . .]*

Qualifiers	**Defaults**
/[NO]DELETE	NODELETE
/LIBRARY=library-spec	/LIBRARY=primary-library
/[NO]LOG	/LOG
/[NO]REPLACE	/REPLACE

Qualifiers

/DELETE
/NODELETE (D)
Deletes an analysis data file after it has been successfully loaded into an SCA library.

/LIBRARY=library-spec
/LIBRARY=primary-library (D)
Specifies an SCA physical library to update. This library must be one of the current SCA libraries, established by a SET LIBRARY command.

If you do not specify this qualifier, SCA refers to the primary SCA library; that is, SCA updates the first of the current SCA physical libraries.

/LOG (D)
/NOLOG
Indicates whether SCA reports successful updating of SCA libraries.

/REPLACE (D)
/NOREPLACE
Indicates whether SCA replaces existing modules of source analysis data with new information.

Parameter

file-spec[, . . .]
Specifies one or more files of source analysis data to be loaded into an SCA library. You may use a wildcard file specification.

The default file type is .ANA, which is the default file type for source analysis data files created by compilers.

Description

With the LOAD command, you can load SCA library files with compiler-generated source information.

Related Commands

SET LIBRARY

Example

```
$ SCA LOAD obj:getfile*
```

Loads the named modules, located at a directory defined as *obj*, into the current library.

For additional examples, see the section about loading a library in the *VAX Language-Sensitive Editor and VAX Source Code Analyzer User Manual.*

LOWERCASE WORD

Changes the letters in the current word or the selected range to lowercase.

Format

LOWERCASE WORD

Description

The LOWERCASE WORD command changes the letters in the current word to lowercase. If the word contains both uppercase and lowercase characters, LSE changes all letters to lowercase.

If the cursor is between words, LSE changes the following word to lowercase. If a selected range is active, all the words within that range are changed to lowercase. The cursor then moves to the start of the next word.

DECwindows Interface Equivalent

Pull-down menu: Format --> Lowercase

Related Commands

CAPITALIZE WORD
UPPERCASE WORD

MODIFY LANGUAGE

Modifies the characteristics of a specified language.

Format

MODIFY LANGUAGE *language-name*

Qualifiers	**Default**
/CAPABILITIES=[NO]DIAGNOSTICS	
/COMMENT=(specifier, . . .)	
/COMPILE_COMMAND=string	
/EXPAND_CASE=AS_IS	
/EXPAND_CASE=LOWER	
/EXPAND_CASE=UPPER	
/FILE_TYPES=(file-type[, . . .])	
/FORTRAN=[NO]ANSI_FORMAT	
/[NO]HELP_LIBRARY=file-spec	
/IDENTIFIER_CHARACTERS=string	
/INITIAL_STRING=string	
/LEFT_MARGIN=*n*	/LEFT_MARGIN=1
/LEFT_MARGIN=CONTEXT_DEPENDENT	
/OVERVIEW_OPTIONS=(MINIMUM_LINES=*m*, TAB_RANGE=(t1,t2))	
/PLACEHOLDER_DELIMITERS= (delimiter-specification[, . . .])	
/PUNCTUATION_CHARACTERS=string	
/[NO]QUOTED_ITEM=(QUOTES=string [,ESCAPES=string])	
/RIGHT_MARGIN=*n*	
/TAB_INCREMENT=*n*	
/TOPIC_STRING=string	
/VERSION=string	
/[NO]WRAP	

Qualifiers

/CAPABILITIES=DIAGNOSTICS
/CAPABILITIES=NODIAGNOSTICS
Specifies whether the compiler can generate diagnostic files.

/COMMENT=(specifier, . . .)
Specifies the character sequences of comments in the language. The specifiers are as follows:

- ASSOCIATED_IDENTIFIER=keyword

  Indicates the preferred association of comments to identifier. You can specify one of the following values:

  - NEXT—Indicates that comments should be associated with the next identifier
  - PREVIOUS—Indicates that comments should be associated with the preceding identifier

- BEGIN=list of quoted strings

  END=list of quoted strings

  Defines the character sequences that start and end bracketed comments. A bracketed comment begins and ends with explicit comment delimiters. (Note that the beginning and ending comment delimiters can be the same, but need not be.) The list provided with the specifiers BEGIN and END can be any of the following:

  - A string that is the one open comment sequence for the language. You must enclose this in quotes.
  - A parenthesized list of strings, each one of which can be an open comment sequence for the language. You must enclose each one in quotes.

  The list accompanying the BEGIN specifier must be consistent with the list acompanying the END specifier. If the BEGIN specifier lists a string, then the END specifier must also list a string.

  Bracketed comments are recognized by the formatting commands (see the ALIGN and FILL commands) and placeholder operations (see the ERASE PLACEHOLDER command and the /DUPLICATION qualifier of the DEFINE PLACEHOLDER command).

- TRAILING=list of quoted strings

  Defines the character sequence that introduces line-oriented comments. A line-oriented comment begins with a special character sequence (consisting of one or more characters) and ends at the end of the line. The list provided with the TRAILING specifier can be any of the following:

  - A string that is the one-line comment sequence for the language.
  - A list of strings enclosed in parentheses; each string can be a line-comment sequence for the language.

  Line comments are recognized by the formatting commands and placeholder operations, just as bracketed comments are.

- LINE=list of quoted strings

  Requires that the comment delimiter be the first character that is not blank on the line. The LINE specifier is particularly useful with block comments, such as the following:

```
/*
** Here is the inside of a comment
** which has LINE="**" specified
*/
```

- FIXED=quoted string, column number

  Used for languages that require that a specific comment delimiter be placed in a specific column, such as FIXED=("*",1) for COBOL.

/COMPILE_COMMAND=string

Specifies the default command string for the COMPILE command. (See the explanation of the *command-string* parameter in the COMPILE command entry.)

/EXPAND_CASE=AS_IS
/EXPAND_CASE=LOWER
/EXPAND_CASE=UPPER

Specifies the case of the text of the inserted template. AS_IS specifies that the inserted template be expanded according to the case in the token or placeholder definition. LOWER and UPPER specify that the inserted template be expanded in lowercase or uppercase, respectively.

/FILE_TYPES=(file-type[, . . .])

Specifies a list of file types that are valid for the language being defined. The file types must be enclosed in quoted strings. When LSE reads a file into a buffer, it sets the language for that buffer automatically if it recognizes the file type. For example, a FORTRAN file type (.FOR) sets the language to FORTRAN. Note that the period character must be included with the file type.

/FORTRAN=ANSI_FORMAT
/FORTRAN=NOANSI_FORMAT

Specifies special processing for ANSI FORTRAN. Note that some commands behave differently when you use the /FORTRAN qualifier. Specifying NOANSI_FORMAT causes LSE to insert templates in non-ANSI (tab) format.

/HELP_LIBRARY=file-spec
/NOHELP_LIBRARY

Specifies the HELP library where you can find help text for placeholders and tokens defined in this language. LSE applies the default file specification SYS$HELP:HELPLIB.HLB. If you want to access some HELP library other than SYS$HELP, you must supply an explicit device name.

/IDENTIFIER_CHARACTERS=string

Specifies the characters that may appear in token and alias names in that language. This list of characters is used in various contexts for the /INDICATED qualifier.

The list of identifier characters also determines what LSE considers to be a word. A word is a sequence of identifier characters, possibly followed by one or more blanks. All nonblank, nonidentifier characters are considered to be distinct words.

If you do not specify the /IDENTIFIER_CHARACTERS qualifier, LSE supplies the following values by default:

```
"abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ%$_0123456789"
```

/INITIAL_STRING=string

Specifies the initial text that is to appear in a newly created buffer.

/LEFT_MARGIN=n
/LEFT_MARGIN=1 (D)
/LEFT_MARGIN=CONTEXT_DEPENDENT
Specifies the left margin setting that is to be associated with the language.

If you specify CONTEXT_DEPENDENT as the column number, then LSE uses the indentation of the current line to determine the left margin when you use the /WRAP qualifier. When you use the FILL command, LSE uses the indentation of the first line of each selected paragraph to determine the left margin.

/OVERVIEW_OPTIONS=(MINIMUM_LINES=m, TAB_RANGE=(t1,t2))
Specifies both the minimum number of lines an overview line must hide and the range of acceptable tab increments.

The specifiers are as follows:

- MINIMUM_LINES=*m*

  Specifies the minimum number of lines an overview line must hide. The default is 1. For example, if the value of the parameter on MINIMUM_LINES is 5, then a line hides other lines only if there are at least five lines to hide. This specifier helps the user to avoid having very small source-line groups, and thus to avoid many expansion levels.

- TAB_RANGE=(t1,t2)

  The TAB_RANGE specifier indicates the range of tab values for which the adjustment definitions are valid. The default is (4,8). The second value must be at least twice the first value; both values must be positive. For example, if the tab range is (4,8), then LSE assumes that the adjustment definitions will work for any DEFINE LANGUAGE/TAB_INCREMENT value from 4 to 8 inclusive. If you specify a /TAB_INCREMENT value outside the tab range, then LSE recomputes indentation to make the adjustments work.

  For best performance, it is recommended that you avoid recomputation by choosing a range that covers reasonable values. The numbers specified for the DEFINE ADJUSTMENT/CURRENT and DEFINE ADJUSTMENT/SUBSEQUENT commands must work for any tab increment value in the tab range.

/PLACEHOLDER_DELIMITERS=(delimiter-specification[, . . .])

Specifies the starting and ending strings that delimit placeholders. Placeholders can specify single constructs or lists of constructs. The delimiters for each type of placeholder are specified as a pair of quoted strings separated by commas and enclosed in parentheses.

The format of a delimiter specification is as follows:

keyword=(starting-string,ending-string)

Possible keywords are REQUIRED, REQUIRED_LIST, OPTIONAL, OPTIONAL_LIST, or PSEUDOCODE. If you do not use the PSEUDOCODE keyword, the default is NOPSEUDOCODE. The maximum length of these strings is seven characters.

The following is an example of a complete set of placeholder delimiter specifications:

```
/PLACEHOLDER_DELIMITERS = ( -
 REQUIRED =("{<",">}"), -
 REQUIRED_LIST=("{<",">}..."), -
 OPTIONAL =("[<",">]"), -
 OPTIONAL_LIST=("[<",">]..."), -
 PSEUDOCODE=("«" , "»"))
```

If any of the five keywords are not specified with the /PLACEHOLDER_DELIMITERS qualifier, LSE applies the following defaults:

```
/PLACEHOLDER_DELIMITERS = ( -
 REQUIRED =("{","}"), -
 REQUIRED_LIST=("{","}..."), -
 OPTIONAL =("[","]"), -
 OPTIONAL_LIST=("[","]..."), -
 NOPSEUDOCODE)
```

/PUNCTUATION_CHARACTERS=string

Specifies the characters that are considered punctuation marks, or delimiters, in the language. When a placeholder name and its enclosing brackets are deleted, preceding white space is also deleted if there are punctuation characters to delimit the program constructs.

/QUOTED_ITEM=(QUOTES=string [,ESCAPES=string])
/NOQUOTED_ITEM

Describes the syntax of certain language elements, such as strings, that require special handling for proper text formatting. LSE uses the

/QUOTED_ITEM qualifier to detect comments properly. LSE does not acknowledge comment strings that occur within quoted items, nor does LSE acknowledge quoted elements that occur within comments.

The value of the /QUOTED_ITEM qualifier indicates the syntax of a quoted item. This value must be a keyword list. The keywords are as follows:

- QUOTES

  This keyword is required, and must have an explicit value. The value must be a quoted string denoting all of the quote characters in the language. LSE assumes that quoted items begin and end with the same character.
- ESCAPES

  This keyword is optional. If given, then the value is required and must be a quoted string containing the escape characters for quoted items. Some languages use escape characters to insert quoting characters into strings. For example, C uses the backslash (\ ) as an escape character. If you omit this keyword, then LSE assumes that the language inserts quote characters into strings by doubling them.

/RIGHT_MARGIN=n

Specifies the right margin setting that is to be associated with the language. By default, the right margin is set at column 80.

/TAB_INCREMENT=n

Specifies that tab stops be set every *n* columns, beginning with column 1.

/TOPIC_STRING=string

Specifies a prefix string to be concatenated to the /TOPIC_STRING qualifier specified in a placeholder or token definition before LSE looks up the help text for that placeholder or token. (Typically, this is the name of the language in the HELP library.)

/VERSION=string

Specifies a string that represents the version number of the tokens and placeholders associated with this language. You use the SHOW LANGUAGE command to display this string.

/WRAP
/NOWRAP
Specifies whether the ENTER SPACE command (bound to the space bar by default) should wrap text when there is too much to fit on the current line. The /NOWRAP qualifier disables such text wrapping.

Parameter

language-name
Specifies the name of the language whose characteristics are to be defined.

Description

With the MODIFY LANGUAGE command, you can supersede text characteristics that you have set for a specific language. It does not affect other characteristics that you may have changed from the initial default by using the DEFINE LANGUAGE command.

Related Commands

DEFINE LANGUAGE
DELETE LANGUAGE
EXTRACT LANGUAGE
SET LANGUAGE
SHOW LANGUAGE

Examples

```
1.  LSE> MODIFY LANGUAGE SAMPLE /EXPAND_CASE=LOWER
```

Makes every letter lowercase in the template for the language *SAMPLE*; this includes the words inside comments.

2.
```
LSE> MODIFY LANGUAGE FORTRAN /FORTRAN=ANSI_FORMAT
```

Sets *ANSI_FORMAT* as the format for your FORTRAN language definition.

3.
```
LSE> MODIFY LANGUAGE Ada /PLACEHOLDER_DELIMITERS=PSEUDOCODE=("«" , "»")
```

Sets pseudocode placeholder delimiters for Ada.

NEXT BUFFER

Moves your next buffer into the current window, returning to your last position in that buffer.

Format

NEXT BUFFER

Description

The NEXT BUFFER command moves the cursor to the next buffer in the list of buffers and maps that buffer to the current window. This allows you to cycle through several buffers without having to type their names.

If you have only two buffers, repeating NEXT BUFFER toggles between them. If you have more than two buffers, the next buffer is determined by the order in which you created the buffers. Only user buffers are included in the list of buffers. For a list of your buffers, use the SHOW BUFFER/USER_BUFFERS command.

If you issue a NEXT BUFFER command while you are positioned in the last buffer in the list, LSE takes you to the first buffer in the list.

Users of the DECwindows interface can press MB1 with the mouse cursor on the buffer name to cycle through the user buffers.

DECwindows Interface Equivalent

Button: User buffer status line --> Buffer name

Related Commands

GOTO BUFFER
PREVIOUS BUFFER
SHOW BUFFER

Example

```
LSE> NEXT BUFFER
```

Moves your next buffer into the current window.

NEXT ERROR

Selects the next diagnostic in the current set of diagnostics.

Format

NEXT ERROR

Description

The NEXT ERROR command positions the cursor at the next diagnostic in the buffer $REVIEW, which contains the current set of diagnostics. If the current error is the last in the set, the NEXT ERROR command does not wrap around from the last error back to the first.

If you are in review mode, a NEXT STEP command is equivalent to a NEXT ERROR command.

DECwindows Interface Equivalent

Pop-up menu: Review buffer --> Next Error
Pull-down menu: Navigate --> Next Error

Related Commands

GOTO REVIEW
NEXT STEP
PREVIOUS ERROR
REVIEW

NEXT OCCURRENCE

Moves the cursor forward to the next occurrence of the current source symbol in the current query and highlights that next occurrence.

Format

NEXT OCCURRENCE

Description

The NEXT OCCURRENCE command moves the cursor forward to the next occurrence in the current query; that occurrence is highlighted. If there are no more occurrences of the current source symbol, LSE interprets the command as a NEXT SYMBOL command. If necessary, LSE remaps the query buffer.

DECwindows Interface Equivalent

Pop-up menu: Query buffer --> Next Occurrence
Pull-down menu: Navigate --> Symbol --> Next Occurrence

Related Commands

NEXT STEP
NEXT SYMBOL
PREVIOUS OCCURRENCE

NEXT QUERY

Moves the cursor to the next SCA query session.

Format

NEXT QUERY

Description

The NEXT QUERY command moves the cursor to the next session in a series of SCA query sessions. LSE maps the query display and moves the cursor to the last remembered position in that query. SCA determines the order of multiple query sessions by the order in which the sessions were created.

DECwindows Interface Equivalent

Button: Query buffer status line --> Query-name

Related Commands

DELETE QUERY
GOTO QUERY
PREVIOUS QUERY

NEXT STEP

Moves the cursor forward to the next error, item, or occurrence, depending on whether LSE is in review or query mode. The indicated item is highlighted.

Format

NEXT STEP

Description

The NEXT STEP command moves the cursor in a manner that depends on the current mode.

- In review mode, LSE treats this command as a NEXT ERROR command.
- In query mode, the NEXT STEP command moves the cursor to the next line in the query display and highlights it, whether it is a symbol or an occurrence.

Keypad Equivalent

Key	Keypad Mode
CTRL/F	All

Related Commands

NEXT ERROR
NEXT NAME
NEXT OCCURRENCE
NEXT SYMBOL
PREVIOUS STEP

NEXT SYMBOL

Moves the cursor forward to the next source symbol in the current query and highlights that next symbol.

Format

NEXT SYMBOL

Description

The NEXT SYMBOL command moves the cursor forward to the next source symbol in the current query and highlights this symbol. If necessary, LSE remaps the query buffer.

DECwindows Interface Equivalent

Pop-up menu: Query buffer --> Next Symbol
Pull-down menu: Navigate --> Symbol --> Next Symbol

Related Commands

NEXT STEP
PREVIOUS SYMBOL

NEXT WINDOW

Moves the cursor from the current window to the next window if the screen is split into multiple windows.

Format

NEXT WINDOW

Description

The NEXT WINDOW command works only if the screen displays multiple windows. LSE positions the cursor in the next window on the screen.

NEXT WINDOW is synonymous with the OTHER WINDOW command.

Keypad Equivalent

Key	Keypad Mode
PF1-↓ NXT WNDW	EDT LK201, EDT VT100
PF1-E6 NXT WNDW	EVE LK201

Related Commands

CHANGE WINDOW_MODE
DELETE WINDOW
ENLARGE WINDOW
ONE WINDOW
OTHER WINDOW
PREVIOUS WINDOW
SET SCREEN
SHRINK WINDOW
TWO WINDOWS

ONE WINDOW

Deletes all but the current window.

Format

ONE WINDOW

Description

The ONE WINDOW command removes from your screen all windows associated with your current editing session, except the one that currently has input focus.

DECwindows Interface Equivalent

Pull-down menu: Display --> One Window

Related Commands

CHANGE WINDOW_MODE
DELETE WINDOW
ENLARGE WINDOW
OTHER WINDOW
PREVIOUS WINDOW
SET SCREEN
SHRINK WINDOW
TWO WINDOWS

OTHER WINDOW

Moves the cursor from the current window to the next window if the screen is split into multiple windows.

Format

OTHER WINDOW

Description

The OTHER WINDOW command works only if the screen displays multiple windows. LSE positions the cursor in the next window on the screen.

The OTHER WINDOW command is synonymous with the NEXT WINDOW command.

Related Commands

CHANGE WINDOW_MODE
DELETE WINDOW
ENLARGE WINDOW
ONE WINDOW
PREVIOUS WINDOW
SET SCREEN
SHRINK WINDOW
TWO WINDOWS

PASTE

Copies the contents of the specified buffer into the current buffer at the current cursor position.

Format

PASTE

Qualifiers	Defaults
/BUFFER=buffer-name	/BUFFER=$PASTE (D)
/CLIPBOARD	See text

Qualifiers

/BUFFER=buffer-name
/BUFFER=$PASTE (D)
Specifies the buffer to be copied into the current buffer.

/CLIPBOARD
Instructs LSE to use the DECwindows clipboard, instead of a buffer, to supply the text being inserted. The /CLIPBOARD and /BUFFER qualifiers are mutually exclusive.

Description

The PASTE command copies text from a specified location to the current buffer. If you do not specify a buffer to copy from, LSE copies from the location (DECwindows Clipboard or character cell terminal $PASTE buffer) that contains the text you last removed using the CUT command.

For users of the DECwindows interface, the default setting is /CLIPBOARD; otherwise, the default is /BUFFER=$PASTE.

Keypad Equivalent

Key	Keypad Mode
PF1-KP6 PASTE	EDT LK201, EDT VT100, EVE LK201
E2 INSERT HERE	EDT LK201, EVE LK201
KP9 INSERT HERE	EVE VT100

DECwindows Interface Equivalent

Pop-up menu: User buffer --> Paste
Pull-down menu: Edit --> Paste

Related Commands

CUT

PREVIOUS BUFFER

Moves your previous buffer into the current window, returning to your last position in that buffer.

Format

PREVIOUS BUFFER

Description

The PREVIOUS BUFFER command moves the cursor back to the previous buffer in the list of buffers and maps that buffer to the current window. This allows you to cycle through several buffers without having to type their names.

If you have only two buffers, repeating PREVIOUS BUFFER toggles between them. If you have more than two buffers, the previous buffer is determined by the order in which you created the buffers. Only user buffers are included in the list of buffers. For a list of your buffers, use the SHOW BUFFER/USER_BUFFERS command.

If you issue a PREVIOUS BUFFER command while you are positioned in the first buffer in the list, LSE takes you to the last buffer in the list.

Related Commands

GOTO BUFFER
NEXT BUFFER
SHOW BUFFER

Example

```
LSE>  PREVIOUS BUFFER
```

Moves your previous buffer into the current window.

PREVIOUS ERROR

Selects the previous diagnostic in the current set of diagnostics.

Format

PREVIOUS ERROR

Description

The PREVIOUS ERROR command positions the cursor at the previous diagnostic in the buffer $REVIEW, which contains the current set of diagnostics. If the current error is the first in the set, the PREVIOUS ERROR command does not wrap around from the first error backwards to the last. If necessary, LSE remaps the $REVIEW buffer.

If you are in review mode, a PREVIOUS STEP command is equivalent to a PREVIOUS ERROR command.

DECwindows Interface Equivalent

Pop-up menu: Review buffer --> Previous Error
Pull-down menu: Navigate --> Previous Error

Related Commands

GOTO REVIEW
NEXT ERROR
PREVIOUS STEP
REVIEW

PREVIOUS OCCURRENCE

Moves the cursor back to the previous occurrence of the current source symbol in the current query and highlights that occurrence.

Format

PREVIOUS OCCURRENCE

Description

The PREVIOUS OCCURRENCE command moves the cursor back to the previous occurrence in the current query; that occurrence is highlighted. If there are no more occurrences of current source symbols, LSE interprets the command as a PREVIOUS ITEM command. If necessary, LSE remaps the query.

DECwindows Interface Equivalent

Pop-up menu: Query buffer --> Previous Occurrence
Pull-down menu: Navigate --> Symbol --> Previous Occurrence

Related Commands

NEXT OCCURRENCE
PREVIOUS ITEM
PREVIOUS STEP

PREVIOUS QUERY

Moves the cursor back to the previous SCA query session.

Format

PREVIOUS QUERY

Description

The PREVIOUS QUERY command moves the cursor back to the previous session in a series of SCA query sessions. LSE maps the query display and moves the cursor to the last remembered position in that query. SCA determines the order of multiple query sessions by the order in which the sessions were created.

Related Commands

DELETE QUERY
GOTO QUERY
NEXT QUERY

PREVIOUS STEP

Moves the cursor back to the previous error, item, name, or occurrence, depending on whether LSE is in review or query mode. That item is highlighted.

Format

PREVIOUS STEP

Description

The PREVIOUS STEP command moves the cursor in a manner that depends on the current mode.

- In review mode, LSE treats this command as a PREVIOUS ERROR command.
- In query mode, the PREVIOUS STEP command moves the cursor to the previous line and highlights it, whether it is a symbol or occurrence.

Keypad Equivalent

Key	**Keypad Mode**
CTRL/B	All

Related Commands

PREVIOUS ERROR
PREVIOUS ITEM
PREVIOUS OCCURRENCE
NEXT STEP

PREVIOUS SYMBOL

Moves the cursor back to the previous source symbol in the current query and highlights that source symbol.

Format

PREVIOUS SYMBOL

Description

The PREVIOUS SYMBOL command moves the cursor back to the previous source symbol in the current query; that source symbol is highlighted. If no more source symbols with the current name exist, LSE interprets the command as a PREVIOUS NAME command. If necessary, LSE remaps the query.

DECwindows Interface Equivalent

Pop-up menu: Query buffer --> Previous Symbol
Pull-down menu: Navigate --> Symbol --> Previous Symbol

Related Commands

NEXT SYMBOL
PREVIOUS STEP

PREVIOUS WINDOW

Moves the cursor from one window to the previous window if the screen is split into multiple windows.

Format

PREVIOUS WINDOW

Description

The PREVIOUS WINDOW command moves the cursor from the bottom window to the top, in sequence, if the screen is split into multiple windows.

Keypad Equivalent

Key	Keypad Mode
PF1-↑ [PRV WNDW]	EDT LK201, EDT VT100
PF1-E5 [PRV WNDW]	EVE LK201

Related Commands

CHANGE WINDOW_MODE
DELETE WINDOW
ENLARGE WINDOW
ONE WINDOW
SET SCREEN
SHRINK WINDOW
TWO WINDOWS

QUIT

Ends an LSE session without saving modified user buffers.

Format

QUIT

Description

The QUIT command ends the editing session without saving modified user buffers.

If you have modified any buffers, LSE warns you that you have changes that will be lost and asks if you want to continue quitting. Typing Y or YES confirms that you want to discard the modified buffers; typing N or NO reactivates the editing session and returns the cursor to the last current buffer.

In DECwindows mode, if you have modified any buffers, LSE displays a dialog box to warn you that modifications will be discarded and to confirm that you want to continue quitting.

DECwindows Interface Equivalent

Pull-down menu: File --> Quit

Related Commands

ATTACH
EXIT
SPAWN

QUOTE

Enters a control code or other character, either as text in the buffer you are editing or as a string for a command.

Format

QUOTE

Description

The QUOTE command enters the character according to the current mode of the buffer, as shown in the status line.

You can also use the QUOTE command for entering strings for search or substitute commands.

If you use the DEFINE KEY command to define a typing key (letter, number, or punctuation mark) or a control key, you can use the QUOTE command to enter the character or control code normally bound to that key.

Keypad Equivalent

Key	**Keypad Mode**
CTRL/V	EDT LK201, EDT VT100, EVE LK201

Related Commands

SET INSERT
SET OVERSTRIKE

Examples

This example shows how you can use the QUOTE command for entering strings for search or substitute commands, as follows:

1. Press the key defined for the SEARCH or SUBSTITUTE command.
2. Press CTRL/V.
3. Press CTRL/J for the line-feed character.

You can define a typing key or a control key and then use the QUOTE command to enter the character or control code normally bound to that key. For example, if you defined the tilde to execute a procedure, you insert a tilde character by doing the following:

1. Press CTRL/V.
2. Type the tilde (~).

READ

Inserts the contents of a file into a buffer.

Format

READ *file-spec*

Qualifier
/BUFFER=buffer-name

Qualifier

/BUFFER=buffer-name
Specifies a buffer into which the file is to be read. If the buffer does not exist, it is created for display only (the buffer cannot be written back to a file).

Parameter

file-spec
Specifies the file to be read. LSE uses the list for the current SET SOURCE_DIRECTORY command to resolve the file specification.

LSE uses VAX DEC/CMS to access a file if the directory for the file to be accessed is the same as the current CMS library.

Description

The READ command opens a file for input and inserts the file's contents into a buffer. LSE inserts the text before the line containing the current position in the receiving buffer; if the buffer previously contained no text, the cursor is positioned at the end of the buffer.

Unless you specify otherwise, the receiving buffer is the current buffer.

READ

Related Commands

GOTO FILE
INCLUDE
SET CMS
SET SOURCE_DIRECTORY
WRITE

Example

```
LSE>  READ x.y
```

Opens file *x.y* for input and reads that file's contents into the current buffer.

RECALL

Recalls a previous LSE command, which you can edit and execute again.

Format

RECALL

Description

The RECALL command recalls a previous LSE command, which you can edit (if necessary) and execute again. You cannot just type RECALL to recall a previous LSE command. If you type RECALL, the command itself is recalled. Instead, use GOLD/DO or a key defined as RECALL.

When you press GOLD/DO, the most recent command you entered reappears in the command window, with the cursor at the end of the command line. To execute the recalled command, you press Return or the Do key.

To recall another command, you press GOLD/DO again, or press the up arrow key (in effect, scrolling back through the command buffer.)

To cancel the recalled command, erase the recalled line (for example, by pressing CTRL/U).

Do not type the command RECALL. If you type RECALL, that command itself is recalled. Instead, use GOLD/DO or a key defined as RECALL.

Keypad Equivalent

Key	Keypad Mode
PF1-DO	EVE LK201, EDT LK201

RECOVER BUFFER

Reconstructs the contents of a buffer from a buffer-change journal file.

Format

RECOVER BUFFER *[file-name]*

Qualifier
/ALL

Qualifier

/ALL
Specifies that LSE should use the latest generation of all locatable buffer-change journal files to attempt to perform a recovery operation. LSE uses the file specification LSE$JOURNAL:.TPU$JOURNAL to locate all buffer-change journal files. If you specify the /ALL qualifier, you cannot specify the *file-name* parameter.

Parameter

file-name
Specifies the name of the file. You can specify either of the following files that the editor should use to perform the recovery operation:

- Source file that was in the buffer
- Full name of the buffer-change journal file

For information about the procedure for recovering changes lost in a system failure, see the section about recovering from a failed editing section in the *VAX Language-Sensitive Editor and VAX Source Code Analyzer User Manual.*

Description

The RECOVER BUFFER command attempts to rebuild the contents of a buffer by using the latest available generation of the file that was in the buffer and a journal file that contains a description of the changes to that buffer. LSE uses the default file specification LSE$JOURNAL:*.TPU$JOURNAL when attempting to locate buffer-change journal files.

Before LSE attempts to recover a buffer, information about the journal file is displayed. When you specify the /ALL qualifier, LSE displays information about each available journal file in succession. You can choose not to recover a buffer if the information describes a journal file other than the one you want.

Related Commands

SET JOURNALING
SET NOJOURNALING

Example

```
LSE>  RECOVER BUFFER login.com
```

Recovers the buffer *LOGIN.COM* from the journal file in LSE$JOURNAL:LOGIN_COM.TPU$JOURNAL.

REFRESH

Refreshes the screen display.

Format

REFRESH

Description

The REFRESH command clears and redisplays the screen, preserving all valid text, including messages in the message window. The cursor returns to its current position.

Keypad Equivalent

Key	Keypad Mode
CTRL/W	All

DECwindows Interface Equivalent

Pull-down menu: Display --> Refresh

Example

```
CTRL/W
```

Causes the screen to go blank for a moment. The display then quickly returns without any extraneous characters that do not belong in your displayed buffers.

REORGANIZE

Optimizes the organization of the specified SCA libraries.

Format

REORGANIZE *[library-spec[, . . .]]*

Qualifier	**Default**
/[NO]LOG	/LOG

Qualifier

/LOG (D)
/NOLOG
Indicates whether SCA reports a successful library reorganization.

Parameter

library-spec[, . . .]
Specifies the SCA libraries to be reorganized. If you do not specify a library, LSE reorganizes the primary SCA library.

Description

The REORGANIZE command optimizes the organization of SCA libraries so that you get the best query and update performance.

Example

```
$ SCA
SCA> CREATE LIBRARY library-directory /MODULE_COUNT=...
SCA> LOAD data-file-directory:*.ANA
SCA> REORGANIZE
```

Creates and optimizes the size and organization of your SCA library.

REPEAT

Repeats a command a specified number of times.

Format

REPEAT *repeat-count command*

Parameters

repeat-count
Specifies a positive decimal integer number indicating the number of times you want to repeat the command.

command
Specifies the command to be repeated.

Description

The REPEAT command repeats a command the number of times you specify.

To repeat a single key, press the PF1 key, followed by one or more keyboard number keys to indicate the number of times you want the key to be repeated. Then, press the key you want.

You cannot use the PF1 key to repeat the delete key or CTRL/Z key.

The repeat operation aborts if you receive a warning of an error while this command is active.

REPEAT

Keypad Equivalent

To repeat a key:

Key	**Keypad Mode**
PF1-number key(s)	All

Examples

1. `LSE> REPEAT 5 ENTER LINE`

   Adds five new lines to the text in the current buffer.

2. $\boxed{\text{PF1}}$ $\boxed{7}$ $\boxed{0}$ $\boxed{=}$

   Inserts 70 equals signs (=) at the current cursor position.

REPLACE

Creates a new generation of the indicated element in your current CMS library.

Format

REPLACE

Qualifier	**Default**
/[NO]VARIANT=variant-letter	/NOVARIANT

Qualifier

/VARIANT=variant-letter
/NOVARIANT (D)
Controls whether CMS creates a variant generation.

Description

The REPLACE command returns to your current CMS library an element name with the same name and type as the input file for your current buffer. When a REPLACE command executes successfully, it creates a new generation of that element; you no longer hold a reservation for the element.

The sequence of actions this command takes are as follows:

1. Writes out the buffer if you have modified it.
2. Performs a CMS REPLACE operation.
3. Deletes the buffer.

REPLACE

DECwindows Interface Equivalent

Pull-down menu: File --> Replace Element

Related Commands

RESERVE
SET CMS
UNRESERVE

Example

```
LSE> REPLACE
```

Creates a new generation of the element with the same name and type as the input file for your current buffer.

REPORT

Produces the specified report.

Format

REPORT *report-name other-parms[. . .]*

Qualifiers	**Defaults**
/DOMAIN=query-name	
/[NO]FILL	/FILL
/HELP_LIBRARY=library_name	
/LANGUAGES=(language,[. . .])	
/OUTPUT=file-name	
/TARGET=target-file-type	See text

Qualifiers

/DOMAIN=query-name
Specifies the name of the query to use as the domain for the report. The query should include occurrences of files that have been compiled. This value is converted to a VAXTPU value and assigned to the global VAXTPU variable SCA$REPORT_DOMAIN_QUERY. This procedure limits the report to objects that are contained, directly or indirectly, within at least one of the files in this query.

The default value is the null string. By convention, VAXTPU report procedures interpret this as the entire SCA library.

/FILL (D)
/NOFILL
Specifies that whenever a paragraph of commented text is inserted into a report, it is set up so that a text processor, such as RUNOFF or DOCUMENT performs the usual fill and justification operations on the paragraph. If you specify /NOFILL, then the report tool does not instruct the text processor to fill or justify the paragraph.

For any individual paragraph, you can override the setting of this qualifier by including appropriate text-processor comments within the body of the comment.

The value of this qualifier is used to set the value of the global VAXTPU variable SCA$REPORT_FILL as follows. If you specify the /FILL qualifier, or it is indicated by default, then SCA$REPORT_FILL is 1; if you specify the /NOFILL qualifier, then SCA$REPORT_FILL is 0.

The /FILL qualifier is ignored if it is not meaningful for the target. In particular, it is ignored for LSE package definitions.

/HELP_LIBRARY=library_name
Specifies the help library to use for PACKAGE reports. This qualifier is ignored for other reports. The PACKAGE report generates one or more DEFINE PACKAGE commands. The *library_name* specifies the value to use with the /HELP_LIBRARY qualifier for the generated DEFINE PACKAGE commands.

If you omit this qualifier, the PACKAGE report omits the /HELP_LIBRARY qualifier from the DEFINE PACKAGE commands it generates.

/LANGUAGES=(language,[. . .])
Specifies the language to use for PACKAGE reports. This qualifier is ignored for other reports. The PACKAGE report generates one or more DEFINE PACKAGES commands. This qualifier specifies the languages to use as the values of the /LANGUAGE qualifier for the generated DEFINE PACKAGE commands.

If you omit this qualifier, the PACKAGE report inserts the LSE placeholder *{language_name}* . . . as the value for the /LANGUAGE qualifier with the DEFINE PACKAGE commands. Before you can execute the DEFINE PACKAGE command, you must replace the placeholder manually with the names of the languages that are appropriate for the languages being defined.

/OUTPUT=file-name
Specifies the output file to use for the report. This value is converted to a VAXTPU string and passed as the value for the global VAXTPU variable SCA$REPORT_OUTPUT. The default value takes the file name from the *report-name* parameter and the file type from the *target-file-type* parameter. The *target-file-type* is implied by the /TARGET qualifier. For example, if you specify DOCUMENT for the /TARGET qualifier, this implies a file type of .SDML.

/TARGET=target-file-type

Specifies the type of target file to produce. This value is converted to a VAXTPU string value and assigned to the global VAXTPU variable SCA$REPORT_TARGET. You can specify one of the following keywords:

Keyword	Type of file
TEXT, TXT	Text file
RUNOFF, DSR, RNO	A file for processing by RUNOFF
SDML, DOCUMENT	A file for processing by VAX DOCUMENT
LSEDIT, LSE	A file for processing by LSE
HLP, HELP	A help file for processing by the VAX Librarian
OTHER=value[1]	Optional file type

[1]The OTHER keyword can take an optional value. The default value is the null string, which by convention is interpreted by the VAXTPU procedures as TEXT. User-supplied report procedures can ignore this convention and provide their own defaults.

The default target file types are SDML for INTERNALS and 2167A_DESIGN reports, HLP for HELP reports, and LSE for PACKAGE reports.

Parameters

report-name

Specifies the name of the report to produce. The command looks for a corresponding VAXTPU procedure by constructing the VAXTPU identifier SCA_REPORT_*report-name* and looking for a VAXTPU procedure with that name to use for producing the report. Because VAXTPU limits identifiers to 132 characters, report names are limited to 132 minus LENGTH"SCA_REPORT_", which equals 121 characters.

Digital has implemented the following reports:

- **HELP**—A help file, suitable for processing by the VMS Librarian into a help library.
- **PACKAGE**—An LSE package definition, which can be processed by LSE and put into an environment file, to create templates for calling the procedures in your code.

- **INTERNALS**—A comprehensive report on the software in your system, all of the information in comment headers, and a structural presentation of your code.
- **2167A_DESIGN**—The design section of the DOD-STD-2167A Software Design Document.

You must type report names completely as they appear in the previous list.

other-parms[. . .]
Specifies other parameters that are passed to the VAXTPU procedure. These parameters are collected into a single string, which is then assigned to the global VAXTPU variable SCA$_REPORT_REST_OF_LINE. These SCA parameters are obtained from the command line from the $REST_OF_LINE type of the VMS Command Definition Utility. For information on the built-in value type $REST_OF_LINE, see the section about defining values in the *VMS Command Definition Utility Manual.*

Description

The REPORT command produces a specified report. For more information about the REPORT command and about customizing reports, see the chapter about customizing reports in the *VAX Language-Sensitive Editor and VAX Source Code Analyzer User Manual.*

The REPORT command requires that LSE be installed even if you are using this command from the SCA command line.

Examples

1.
```
LSE> REPORT HELP /TARGET=HELP
```

   Produces a report named *HELP* with a file type of .HLP for processing with the VAX Librarian.

2.
```
SCA> FIND/NAME=abc_files abc* AND symbol=file AND occ=command_line
SCA> REPORT/DOMAIN=abc_files INTERNALS
```

   Produces an INTERNALS report only on files with names beginning with ABC.

RESERVE

Reserves an element in your current CMS library.

Format

RESERVE *[element-name]*

Qualifiers	**Default**
/GENERATION[=generation-exp]	
/[NO]MERGE=generation-exp	/NOMERGE

Qualifiers

/GENERATION[=generation-exp]
Specifies the generation of the element to reserve. If you do not specify a value, LSE assumes you have specified generation "1+" (the *generation-exp* parameter must be enclosed in quotation marks if nonalphanumeric characters are present). If you omit the qualifier altogether, LSE uses the specified or default value from the command SET CMS/GENERATION to determine the generation to reserve.

/MERGE=generation-exp
/NOMERGE (D)
Determines whether LSE merges another generation of the element with the generation being reserved. If you omit this qualifier, LSE uses the setting of the command SET CMS/[NO]MERGE to determine whether to merge generations of the element being reserved.

You must enclose *generation-exp* in quotes (" ") if nonalphanumeric characters are present.

RESERVE

Parameter

element-name
Specifies the elements to reserve. If you do not specify an element name, LSE uses the file name and type of your current buffer as the element name.

Description

The RESERVE command executes the CMS command RESERVE on the indicated element in your current CMS library and reads the file created into the current editing buffer.

To specify conditions for reserving the element, use the SET CMS command with its available command qualifiers.

DECwindows Interface Equivalent

Pull-down menu: File --> Reserve Element

Related Commands

REPLACE
SET CMS
UNRESERVE

Example

```
LSE>  RESERVE USER.TXT
```

Reserves an element called *USER.TXT* in your current CMS library.

REVIEW

Selects and displays a set of diagnostic messages that resulted from a compilation. The diagnostics associated with the current contents of the buffer become the current diagnostic set.

Format

REVIEW *[buffer]*

Qualifier
/FILE=file-spec

Qualifier

/FILE=file-spec
Specifies the name of the diagnostics file containing the results of a compilation. By default, LSE looks in your current directory for a .DIA file with the same file name as the file associated with the buffer.

Parameter

buffer
Specifies that the set of diagnostics associated with the named buffer is to be reviewed. The default is the current buffer.

Description

The REVIEW command selects and displays a set of diagnostic messages associated with the current contents of a buffer.

A set of diagnostics becomes associated with a buffer by a COMPILE/REVIEW command or a REVIEW command. It remains associated with that buffer until you issue a subsequent COMPILE command for that buffer, a REVIEW command with an explicit /FILE qualifier, or an END REVIEW command.

If no diagnostics are associated with the buffer, LSE attempts to read a set of diagnostics from a file. If you do not supply a file specification, LSE uses the name of the file associated with the buffer, but with .DIA as the file type. You may use the /FILE qualifier to override this default.

You may use the REVIEW command at any time to change the set of diagnostics to be reviewed. If you use the REVIEW command to return to a set of diagnostics, the last diagnostic and region selected in that set become the current diagnostic and region.

DECwindows Interface Equivalent

Pull-down menu: File --> Review

Related Commands

COMPILE/REVIEW
END REVIEW
GOTO REVIEW

Example

```
LSE> REVIEW
```

Displays compilation diagnostics in a window containing the $REVIEW buffer, after you have used the /DIAGNOSTICS qualifier to invoke a compiler.

SAVE ENVIRONMENT

Writes out all user-defined languages, placeholders, tokens, aliases, and packages to an environment file.

Format

SAVE ENVIRONMENT *file-spec*

Qualifiers	**Defaults**
/ALL	/ALL
/NEW	/ALL

Qualifiers

/ALL (D)
Specifies that LSE write all defined items to the environment file.

/NEW
Specifies that LSE write out only those definitions you made during the current editing session. Definitions that were read in from an environment file are not written.

Parameter

file-spec
Specifies the file to which LSE should write the environment data.

SAVE ENVIRONMENT

Description

The SAVE ENVIRONMENT command writes out all user-defined languages, placeholders, tokens, aliases, and packages to an environment file. This procedure saves processing time when LSE reads the definitions back in. (See the section about using environment and section files in the *VAX Language-Sensitive Editor and VAX Source Code Analyzer User Manual* for a discussion of the use of the logical name LSE$ENVIRONMENT or the LSE command line qualifier /ENVIRONMENT to restore definitions in an environment file.)

Usually, LSE writes all user-defined items to the environment file. You may supply user-defined items with the LSEDIT /INITIALIZATION and /ENVIRONMENT qualifiers, or with DEFINE commands during the editing session. You can use the /NEW qualifier to tell LSE to write only those items defined during the current editing session.

DECwindows Interface Equivalent

Pull-down menu: Customize --> Save Current Attributes

Related Commands

SAVE SECTION

Example

```
LSE>  SAVE ENVIRONMENT myfile.env
```

Creates an environment file named *myfile.env* to hold any current language, placeholder, token, alias, and package definitions.

SAVE SECTION

Writes the binary form of all current key definitions, learn sequences, and VAXTPU procedures and variables to a section file. This saves processing time when LSE reads the definitions back in.

Format

SAVE SECTION *file-spec*

Qualifiers	**Defaults**
/[NO]DEBUG_NAMES	/DEBUG_NAMES
/IDENT=string	
/[NO]PROCEDURE_NAMES	/PROCEDURE_NAMES

Qualifiers

/DEBUG_NAMES (D)
/NODEBUG_NAMES
Specifies whether VAXTPU procedure parameters or local variable names should be written to the section file.

/IDENT=string
Specifies an identifying string for the section file.

/PROCEDURE_NAMES (D)
/NOPROCEDURE_NAMES
Specifies whether VAXTPU procedure names should be written to the section file.

Parameter

file-spec
Specifies the file to which LSE should write the section data. The default file type is .TPU$SECTION.

SAVE SECTION

Description

The SAVE SECTION command writes key definitions, learn sequences, user-defined commands, mode settings, VAXTPU procedures, and VAXTPU variable names to a section file so that they may be restored at a later time. (See the section about using environment and section files in the *VAX Language-Sensitive Editor and VAX Source Code Analyzer User Manual* for a discussion of the use of the logical name LSE$SECTION or the LSE command line qualifier /SECTION to restore definitions saved in the section file.)

The SAVE SECTION command calls the VAXTPU built-in SAVE procedure to actually write the section file. By default, the type of the saved section file is .TPU$SECTION.

Related Commands

SAVE ENVIRONMENT

Example

```
LSE>  SAVE SECTION MY_SECTION
```

Creates a section file named MY_SECTION.TPU$SECTION; the file saves all current key definitions, learn sequences, VAXTPU procedures, and variable names.

SEARCH

Searches the current buffer for the specified string and positions the cursor at that string.

Format

SEARCH *search-string*

Qualifiers	**Defaults**
/DIALOG	/NODIALOG
/[NO]PATTERN	/NOPATTERN

Qualifiers

/DIALOG
/NODIALOG (D)
Instructs LSE to use a dialog box to prompt the user for parameters and qualifier values. The command parameters are optional if you specify this qualifier. If you supply command parameters and qualifiers with the /DIALOG qualifier, these parameters and qualifiers are used to set the initial state of the dialog box.

LSE ignores the /DIALOG qualifier if you are using a character-cell terminal.

/PATTERN
/NOPATTERN (D)
Enables or disables special interpretation of wildcard characters and a quote character in the *search-string* parameter. You can set the syntax for specifying a pattern to the VMS style or ULTRIX style. Table CD–3 lists the VMS-style wildcards. Table CD–4 lists the ULTRIX-style wildcards.

Table CD–3: VMS-Style Wildcards

Wildcard	Matches
*	One or more characters of any kind on a line.
**	One or more characters of any kind crossing lines.
%	A single character.
\<	Beginning of a line.
\>	End of a line.
\[set-of-characters]	Any character in the specified set. For example, \[abc] matches any letter in the set "abc" and \[c-t] matches any letter in the set "c" through "t."
\[~set-of-characters]	Anything *not* in the specified set of characters.
\	Lets you specify the characters \,*,% or] within wildcard expressions. For example, \\ matches the backslash character (\ ).
\.	Repeats the previous pattern zero or more times, including the original.
\:	Repeats the previous pattern at least once, including the original; that is, a null occurrence does not match.
\w	Any empty space created by the space bar or tab stops, including no more than one line break.
\d	Any decimal digit.
\o	Any octal digit.
\x	Any hexadecimal digit.
\a	Any alphabetic character, including accented letters, other marked letters, and non-English letters.
\n	Any alphanumeric character.
\s	Any character that can be used in a symbol: alphanumeric, dollar sign, and underscore.
\l	Any lowercase letter.
\u	Any uppercase letter.

(continued on next page)

Table CD–3 (Cont.): VMS-Style Wildcards

Wildcard	Matches
\p	Any punctuation character.
\f	Any formatting characters: backspace, tab, line feed, vertical tab, form feed, and carriage return.
\^	Any control character.
\+	Any character with bit 7 set; that is, ASCII decimal values from 128 through 255.

Table CD–4: ULTRIX-Style Wildcards

Wildcard	Matches
.	A single character.
^	Beginning of a line.
$	End of a line.
[set-of-characters]	Any character in the specified set. For example, [abc] matches any letter in the set "abc" and [c-t] matches any letter in the set "c" through "t."
[^set-of-characters]	Anything *not* in the specified set of characters.
\	Lets you specify the characters \,.,^,$,[,],or * in wildcard expressions. For example, \\ matches the backslash character (\ ).
*	Repeats the previous pattern zero or more times, including the original.
+	Repeats the previous pattern at least once, including the original; that is, a null occurrence does not match.

When you specify the /NOPATTERN qualifier (or when it is the default), special interpretation of the asterisk, percent sign, and backslash characters is disabled.

SEARCH

Parameter

search-string
Specifies a quoted string indicating the string to search for.

If you are using the DECwindows interface and specify the /DIALOG qualifier, the search string field in the Find dialog box takes the default value from the the previous search string, if any.

Description

The SEARCH command searches the current buffer in the specified direction for the specified character string, but ignores any occurrence of the search string that begins at the current cursor position. If the search is successful, LSE positions the cursor on the first character of the string. If LSE does not find the string, it issues a message indicating that no matching string was found.

The direction in which a search is performed is independent of the current direction set for a buffer. This lets you change the direction of the search operation without changing the current direction set for the buffer. The prompts for the search string reflect this behavior. Note that you can change the direction of the search by pressing a key that changes the search direction; this can be the first key you press in response to the prompt, or the key that terminates the prompt.

When conducting a search, LSE regards uppercase and lowercase letters as equivalent. To alter this behavior, see the SET SEARCH command.

If you specify a null string as the search string, LSE searches for the last search string given in the SEARCH command. If LSE prompts you for a search string, you must not use quotation marks in your response, unless you want LSE to search for a string that includes quotation marks.

The direction in which LSE executes the SEARCH command is determined by the key used to end the SEARCH command. If you end your response to the prompt with a keypad key bound to SET FORWARD or SET REVERSE, LSE changes the search direction before the SEARCH command. This is not true in DECwindows if you are specifying search strings through the dialog box.

Keys bound to other commands end the string and LSE conducts the search in the current direction.

For information about searching for a formatting or control character, see the QUOTE command.

Keypad Equivalent

SEARCH

Key	Keypad Mode
PF1-PF3 FIND	EDT LK201, EDT VT100, EVE LK201
E1 FIND	EDT LK201, EVE LK201
KP4 FIND	EVE VT100

SEARCH ""

Key	Keypad Mode
PF3 FNDNXT	EDT LK201, EDT VT100, EVE LK201

SEARCH/PATTERN

Key	Keypad Mode
PF1-E1 FNDPATT	EDT LK201, EVE LK201

DECwindows Interface Equivalent

SEARCH/DIALOG
Pull-down menu: Navigate --> Find . . .

SEARCH ""
Pop-up menu: User buffer --> Find Next
Pull-down menu: Navigate --> Find Next

Related Commands

QUOTE
SET SEARCH
SHOW SEARCH

Examples

1.
```
LSE>  SEARCH "the editor"
```

Searches the current buffer for the next occurrence of the string *the editor*. The quotation marks in the search string indicate to LSE that you are searching for the words enclosed in the quotation marks.

2. FIND
```
_Forward Search:  open
```

Searches the current buffer for the next occurrence of the word *open*.

3.
```
LSE>  SEARCH/PATTERN "2%\%"
```

Searches the current buffer for the next occurrence of a string consisting of the number 2, any character, and a percent sign. Text that would satisfy this condition includes the strings "20%" and "29%."

SELECT ALL

Selects the entire contents of the current buffer.

Format

SELECT ALL

Description

The SELECT ALL command places all the contents of the current buffer in the selected range. Any operations that LSE performs on a selected range then apply to all the contents of the buffer.

DECwindows Interface Equivalent

Pull-down menu: Edit --> Select all

SET AUTO_ERASE

Enables automatic erasing of placeholders in the indicated buffer.

Format

SET AUTO_ERASE

Qualifier
/BUFFER=buffer-name

Qualifier

/BUFFER=buffer-name
Indicates the buffer for which automatic erasing is to be enabled. The default is the current buffer.

Description

The SET AUTO_ERASE command enables LSE to erase the placeholder that the cursor is on when you type a character over that placeholder in the indicated buffer. However, if the cursor is on the first character of an open placeholder delimiter, LSE displays the characters you type without erasing the placeholder.

Initially, LSE is set to automatically erase placeholders.

Related Commands

SET NOAUTO_ERASE
SHOW BUFFER

Example

```
LSE>  SET AUTOERASE/BUFFER=USER.TXT
```

Enables automatic erasing of placeholders in the buffer *USER.TXT*.

SET CMS

Sets the default values for reservations and fetches that LSE performs when you issue the appropriate LSE file-manipulation commands.

Format

SET CMS

Qualifiers
/[NO]CONCURRENT
/[NO]CONFIRM
/GENERATION=generation-exp
/[NO]HISTORY
/[NO]MERGE=generation-exp
/[NO]NOTES
/[NO]REMARK=string

Qualifiers

/CONCURRENT
/NOCONCURRENT
Controls whether an element reserved by you can be reserved by another user while you have it reserved. The initial setting is /CONCURRENT.

/CONFIRM
/NOCONFIRM
Specifies whether you want to be prompted for confirmation before LSE performs a FETCH or RESERVE operation. The initial setting is /CONFIRM.

/GENERATION=generation-exp
Specifies the generation to be used for CMS RESERVE and FETCH operations. The initial setting is /GENERATION="1+".

/HISTORY
/NOHISTORY
Controls whether CMS includes the element history in the file if the element has the history attribute and if a CMS FETCH or CMS RESERVE operation is performed. The initial setting is /HISTORY.

/MERGE=generation-exp
/NOMERGE
Controls whether LSE merges a reserved or fetched element with another generation of the same element. The initial setting is /NOMERGE.

/NOTES
/NONOTES
Controls whether notes are embedded in the file if the retrieved element has the notes attribute and if a CMS FETCH or CMS RESERVE operation is performed. The initial setting is /NOTES.

/REMARK=string
/NOREMARK
Specifies the remark to be used on RESERVE operations. The initial setting is to prompt for the remark. If you specify the /NOREMARK qualifier, LSE prompts you for a remark when you issue a CMS file-manipulation command.

Description

The SET CMS command specifies default settings for LSE's file-manipulation commands that reserve or fetch files.

The effect of the SET CMS command is cumulative; that is, issuing a SET CMS/NOHISTORY command followed by a SET CMS/NONOTES command causes both /NOHISTORY and /NONOTES to be set. (You would then need to issue the command SET CMS/HISTORY to set /HISTORY again.)

If you do not specify any qualifiers, the SET CMS command resets all values to their initial settings.

NOTE

The SET CMS command settings are not used by any commands that begin with the word CMS.

SET CMS

DECwindows Interface Equivalent

Pull-down menu: Customize --> CMS Attributes . . .

Related Commands

GOTO FILE
GOTO SOURCE
READ
REPLACE
RESERVE
SHOW CMS
UNRESERVE

Example

```
LSE>  SET CMS/GENERATION=Baselevel_1
```

Causes fetches performed by the commands GOTO FILE, GOTO SOURCE, and READ to use the generation that corresponds to the class *Baselevel_1*. Any reservations made using the RESERVE command also use this class.

SET CURSOR

Selects either bound cursor motion or free cursor motion.

Format

SET CURSOR *motion-setting*

Parameter

motion-setting
Specifies the cursor-motion setting. Motion-setting keywords and their effects are as follows:

BOUND
Restricts the cursor to positioning on a character, end-of-line, or end-of-buffer. This is the initial setting and is similar to cursor motion in the EDT editor.

FREE
Lets the cursor move anywhere in a window including past the end-of-line, past the end-of-buffer, in the middle of a tab, or to the left of the left margin. This is similar to the default cursor motion for the EVE editor.

Description

The SET CURSOR command either binds the cursor to that part of the buffer occupied by text, or sets it free to be positioned anywhere in the buffer, depending on the parameter you specify.

SET CURSOR

DECwindows Interface Equivalent

Pull-down menu: Customize --> Global Attributes . . .

Related Commands

SHOW MODE

SET DEFAULT_DIRECTORY

Changes your default device and directory specifications.

Format

SET DEFAULT_DIRECTORY *[device-name[:]][directory-spec]*

Parameters

device-name[:]
Specifies a device name to be used as the default device in a file specification.

directory-spec
Specifies a directory name to be used as the default directory in a file specification. A directory name must be enclosed in brackets. Use the minus sign to specify the next higher directory from the current default directory.

You must specify either the *device-name* parameter or the *directory-spec* parameter. If you specify only the device name, the current directory is the default for the *directory-spec* parameter. If you specify only the directory name, the current device is the default for the *device-name* parameter.

You can use a logical name, but it must constitute at least the device part of the specification.

Description

The SET DEFAULT_DIRECTORY command changes your default device and directory names, along with any equivalence strings. The new default is applied to all subsequent file specifications that do not explicitly include a device or directory name.

The default set in an LSE editing session remains in effect after you terminate the LSE session.

SET DEFAULT_DIRECTORY

Related Commands

SHOW DEFAULT_DIRECTORY

Example

```
LSE>  SET DEFAULT DISK$:[USER.LSE]
```

Establishes *DISK$:[USER.LSE]* as the default directory for LSE to use in accessing files.

SET DIRECTORY

Sets the default read-only/writeable status of files in a specified directory.

Format

SET DIRECTORY *directory-spec*

Qualifiers	**Defaults**
/READ_ONLY	/WRITE
/WRITE	/WRITE

Qualifiers

/READ_ONLY
Specifies that files in the indicated directories are read-only and unmodifiable by default. The /READ_ONLY qualifier prevents the WRITE command from writing files to the indicated directory unless you subsequently override this default.

/WRITE (D)
Specifies that files in the indicated directories are writeable and unmodifiable by default.

Parameter

directory-spec
Specifies a directory to be set as read-only or writeable.

Description

The SET DIRECTORY command determines the read-only/writeable status of a directory you specify. The logical name LSE$READ_ONLY_DIRECTORY stores the list of read-only directories.

SET DIRECTORY

Related Commands

SHOW DIRECTORY

Example

```
LSE>  SET DIRECTORY/READ_ONLY [LIBRARY_DIRECTORY]
```

Specifies files in the directory *LIBRARY_DIRECTORY* as unmodifiable.

SET FONT

Sets the specified fonts for the screen.

Format

SET FONT *keyword-list*

Parameter

keyword-list
Indicates the fonts to be set or reset. The types of fonts are as follows:

BIG
Specifies that the fonts should be big.

CONDENSED
Specifies that the fonts should be condensed.

LITTLE
Specifies that the fonts should be little.

NORMAL
Specifies that the fonts should be normal.

Description

The SET FONT command sets the fonts to big or little, normal or condensed. You can specify either big or little and either normal or condensed.

You use the SET FONT command only with DECwindows.

SET FONT

DECwindows Interface Equivalent

Pull-down menu: Customize --> Window Attributes . . .

Related Commands

SHOW SCREEN

Example

```
LSE>  SET FONT BIG,CONDENSED
```

Set the fonts to big and condensed.

SET FORWARD

Sets the current direction of a buffer to forward.

Format

SET FORWARD

Qualifier
/BUFFER=buffer-name

Qualifier

/BUFFER=buffer-name
Indicates the buffer whose direction is to be set to forward. The default is the current buffer.

Description

The SET FORWARD command sets the current direction of the indicated buffer to forward. The status line of each buffer displays the current direction.

Users of the DECwindows interface can switch direction by selecting the status line button and pressing MB1.

Keypad Equivalent

Key	Keypad Mode
KP4 FORWARD	EDT LK201, EDT VT100, EVE LK201
None	EVE VT100

SET FORWARD

DECwindows Interface Equivalent

Button: Buffer status line $\left\{ \begin{array}{l} \text{Forward} \\ \text{Reverse} \end{array} \right\}$

Related Commands

CHANGE DIRECTION
SET REVERSE

SET INDENTATION

Sets the current indentation level count for the current buffer, without changing the current line.

Format

SET INDENTATION *level-option*

Qualifier
/BUFFER=buffer-name

Qualifier

/BUFFER=buffer-name
Indicates the buffer whose current indentation level count is to be changed. The default is the current buffer.

Parameter

level-option
Indicates the level to be set or changed. The indentation keywords and their effects are as follows:

CURRENT
Sets the indentation level count to the beginning of the text on the current line.

CURSOR
Sets the indentation level count to the column currently occupied by the cursor.

LEFT
Decreases the indentation level count by the current tab increment.

RIGHT
Increases the indentation level count by the current tab increment.

Description

The SET INDENTATION command sets the current indentation level count for the current buffer. A TAB or ENTER TAB command given at the beginning of a line inserts tabs and blanks corresponding to the current indentation level count.

Related Commands

CHANGE INDENTATION
ENTER TAB
EXPAND
TAB
UNTAB

SET INSERT

Sets the text-entry mode of the indicated buffer to insert mode.

Format

SET INSERT

Qualifier
/BUFFER=buffer-name

Qualifier

/BUFFER=buffer-name
Indicates the buffer whose text-entry mode is to be changed. The default is the current buffer.

Description

The SET INSERT command sets the mode of the indicated buffer to insert. In insert mode, LSE inserts typed characters before the current cursor position.

The status line of each buffer displays the current text-entry mode.

Users of the DECwindows interface can cycle through Insert, Overstrike, and Nomodify by selecting the status line button and pressing MB1.

DECwindows Interface Equivalent

$$\textbf{Button}\text{: Buffer status line} \left\{ \begin{array}{l} \text{Insert} \\ \text{Overstrike} \\ \text{Nomodify} \end{array} \right\}$$

Related Commands

CHANGE TEXT_ENTRY_MODE
SET OVERSTRIKE

SET JOURNALING

Enables buffer-change journaling for the specified buffers.

Format

SET JOURNALING *[buffer-name]*

Qualifier
/ALL

Qualifier

/ALL
Specifies that all of LSE's user buffers that exist when the command is issued should be journaled. If you specify the /ALL qualifier, you cannot specify the *buffer-name* parameter.

Parameter

buffer-name
Specifies the name of the buffer that should be journaled. If you omit this parameter, the default is the current buffer.

Description

The SET JOURNALING command starts buffer-change journaling for the specified user buffer. SET JOURNALING does not allow buffer-change journaling for system buffers.

Related Commands

RECOVER BUFFER
SET NOJOURNALING

Example

```
LSE> SET JOURNALING login.com
```

Enables buffer-change journaling for the buffer *login.com*. Buffer changes are written to the file LSE$JOURNAL:LOGIN_COM.TPU$JOURNAL.

SET LANGUAGE

Sets the language associated with the indicated buffer.

Format

SET LANGUAGE *language-name*

Qualifier
/BUFFER=buffer-name

Qualifier

/BUFFER=buffer-name
Indicates the buffer whose associated language you want to set. The current buffer is the default.

Parameter

language-name
Specifies the name of the language to associate with the buffer. The SET LANGUAGE command requires this parameter.

Description

The SET LANGUAGE command associates a language with a buffer. By default, LSE uses a file-type specification to determine the language to associate with the buffer. If LSE cannot determine the language from the file type, or if no file is associated with the buffer, LSE uses the language in effect when you created the buffer. If you attempt to associate a language with a system buffer, such as $REVIEW, $MESSAGES, or $HELP, you receive an error message.

To disassociate a language with a specified buffer, use the SET NOLANGUAGE command.

SET LANGUAGE

Related Commands

DEFINE LANGUAGE
DELETE LANGUAGE
SET NOLANGUAGE
SHOW LANGUAGE

Example

```
LSE>  SET LANGUAGE example
```

Associates the language *example* with the current buffer.

SET LEFT_MARGIN

Sets the left margin for the indicated buffer.

Format

SET LEFT_MARGIN *column-number*

Qualifier
/BUFFER=buffer-name

Qualifier

/BUFFER=buffer-name
Indicates the buffer whose left margin is to be changed. The default is the current buffer.

Parameter

column-number
Specifies the column for the left margin. The value must be greater than or equal to one and less than the value set for the right margin.

If you specify the CONTEXT_DEPENDENT value as the column number, then LSE uses the indentation of the current line to determine the left margin when you use the /WRAP qualifier. When you use the FILL command, LSE uses the indentation of the first line of each selected paragraph to determine the left margin.

SET LEFT_MARGIN

Description

The SET LEFT_MARGIN command sets the left margin for a buffer. The FILL and ENTER LINE commands use this margin setting. By default, the left margin is at column 1.

To find out the setting of the left margin, use the SHOW BUFFER command.

Related Commands

SET RIGHT_MARGIN
SHOW BUFFER [DECwindows interface]

Example

```
LSE> SET LEFT_MARGIN 10
```

Sets the left margin in the current buffer at column 10.

SET LIBRARY

Identifies the SCA physical libraries to be used for subsequent SCA functions.

Format

SET LIBRARY *directory-spec[, . . .]*

Qualifiers	**Default**
/AFTER[=library-spec]	
/BEFORE[=library-spec]	
/[NO]LOG	/LOG

Qualifiers

/AFTER[=library-spec]
Instructs SCA to insert the new library or libraries into the list of active SCA libraries after the library you specify as the qualifier value. If you do not specify a value, SCA adds the library or libraries to the end of the list.

/BEFORE[=library-spec]
Instructs SCA to insert the new library or libraries into the list of active SCA libraries before the library you specify as the qualifier value. If you do not specify a value, SCA adds the library or libraries to the beginning of the list.

/LOG (D)
/NOLOG
Indicates whether SCA reports the resulting list of active SCA libraries.

SET LIBRARY
SCA Command

Parameter

directory-spec[, . . .]
Specifies one or more directories, each of which comprises a separate SCA library. The list of libraries you specify replaces the current list of active libraries unless you specify an /AFTER or /BEFORE qualifier.

Description

The SET LIBRARY command lets you activate the specified library for use during the current SCA session. If you list several directories, SCA can access all of them during your session as a single logical library. When you subsequently invoke SCA, it uses the logical name SCA$LIBRARY to reestablish the active library list.

Related Commands

SET NOLIBRARY

Example

```
$ SCA SET LIBRARY DISK$:[USER.SCALIB]
```

Defines the library named as the one SCA uses for subsequent access.

See the chapter about using SCA libraries in the *VAX Language-Sensitive Editor and VAX Source Code Analyzer User Manual* for additional examples.

SET MARK

Associates a marker name with the current cursor position. You can later use that marker name with the GOTO MARK command to return to the specified position.

Format

SET MARK *marker-name*

Parameter

marker-name
Specifies the name of the marker to be placed. For a marker name, you may use any combination of up to 21 alphanumeric characters, underscores, or dollar signs. If this marker name is already in use, the previous marker is canceled.

Description

The SET MARK command tells LSE to remember the current cursor position by a marker. The command is useful if you are editing a large file and want to go back to a particular point in the text without having to search through the file.

Related Commands

GOTO MARK

Example

```
LSE>  SET MARK M
```

Sets a marker named *M* as the reference for the current cursor position. Thereafter, issuing the command GOTO MARK M returns the cursor to this position.

SET MODE

Establishes the status of warning bells sounding, keypad emulation, select range, tab appearance, and the use of graphic characters in menus.

Format

SET MODE *keyword-list*

Parameter

keyword-list
Indicates the modes to be set or reset. The mode keywords and their effects are as follows:

BELL=NONE
BELL=ALL
BELL=BROADCAST
BELL=NOBROADCAST
Specifies which new messages should be accompanied by a warning bell character. By default, only broadcast messages are accompanied by a warning bell.

KEYPAD=EDT
KEYPAD=EVE
Specifies whether the key definitions should be similar to EDT or EVE. Note that EVE key definitions do not use the numeric keypad on VT200 terminals; numeric keypads on VT200-series terminals emulate EDT key definitions regardless of the keypad mode you choose.

MENU=[NO]GRAPHICS
Lets you choose between graphic characters and nongraphic characters in the display of a menu. The initial setting is MENU=GRAPHICS. If the terminal characteristics do not include DEC_CRT, then LSE uses nongraphic characters regardless of the setting of this mode.

Graphic characters currently require more screen repainting than do nongraphic characters, so you may want to use SET MODE MENU=NOGRAPHICS if you are working at a low baud rate.

PENDING_DELETE
NOPENDING_DELETE
Specifies whether a selection in a user buffer should be deleted when the user inserts text. The initial setting is NOPENDING_DELETE. PENDING_DELETE is disabled for a selection made with SELECT ALL. You can use the UNERASE SELECTION command to restore deleted text.

TAB=VISIBLE
TAB=INVISIBLE
Specifies whether tabs should appear as blanks, or a combination of the HT (horizontal tab) symbol and dots ("HT......").

Description

The SET MODE command establishes the status of warning bells sounding, keypad emulation, selected range, tab appearance, and the use of graphic characters.

DECwindows Interface Equivalent

Pull-down menu: Customize --> Global Attributes . . .

Related Commands

SHOW MODE

Examples

1.
```
LSE>  SET MODE BELL=NOBROADCAST
```

   Prevents the warning bell from sounding when broadcast messages appear in the LSE message buffer.

2.
```
LSE>  SET MODE KEYPAD=EVE
```
Sets key definitions to be the same as those used with EVE.

3.
```
LSE>  SET MODE PENDING_DELETE
```
Causes a selection to be deleted when the user inserts text into a user buffer.

4.
```
LSE>  SET MODE TAB=INVISIBLE
```
Causes tabs to appear as blanks.

SET MODIFY

Sets buffer status to modifiable.

Format

SET MODIFY

Qualifier
/BUFFER=buffer

Qualifier

/BUFFER=buffer
Indicates the buffer that is to be set modifiable. The current buffer is the default.

Description

The SET MODIFY command changes the status of the current buffer, or the buffer specified, from unmodifiable to modifiable.

Users of the DECwindows interface can cycle through Insert, Overstrike, and Nomodify by selecting the status line button and pressing MB1. If the status line shows Insert and Overstrike, then the buffer is modifiable.

DECwindows Interface Equivalent

Button: Buffer status line $\left\{\begin{array}{l}\text{Insert}\\ \text{Overstrike}\\ \text{Nomodify}\end{array}\right\}$

Related Commands

SET CMS
SET DIRECTORY
SET NOMODIFY

Example

```
LSE> SET MODIFY
```

Allows you to modify a file that you previously brought into the current buffer as display-only.

SET NOAUTO_ERASE

Disables automatic erasing of placeholders in the indicated buffer.

Format

SET NOAUTO_ERASE

Qualifier
/BUFFER=buffer-name

Qualifier

/BUFFER=buffer-name
Indicates the buffer for which automatic erasing is to be disabled. The default is the current buffer.

Description

The SET NOAUTO_ERASE command prevents LSE from automatically erasing the placeholder that the cursor is on when you type a character over that placeholder in the indicated buffer.

Initially, LSE is set to automatically erase placeholders.

Related Commands

SET AUTO_ERASE
SHOW BUFFER [DECwindows interface]

Example

```
LSE> SET NOAUTOERASE/BUFFER=USER.TXT
```

Disables automatic erasing of placeholders in the buffer *USER.TXT*.

SET NOJOURNALING

Disables buffer journaling for the specified buffers.

Format

SET NOJOURNALING *[buffer-name]*

Qualifier
/ALL

Qualifier

/ALL
Specifies that all of LSE's buffer-change journal files should be closed and buffer-change journaling halted for those buffers. If you specify the /ALL qualifier, you cannot specify the *buffer-name* parameter.

Parameter

buffer-name
Specifies the name of the buffer that no longer have an associated buffer-change journal file. If you omit this parameter, the default is the current buffer.

Description

The SET NOJOURNALING command terminates buffer-change journaling for the specified buffer. Any subsequent changes to the buffer are not journaled unless you use the SET JOURNALING command to enable buffer-change journaling.

Related Commands

RECOVER BUFFER
SET JOURNALING

Example

```
LSE> SET NOJOURNALING login.com
```

Terminates buffer-change journaling for the buffer *login.com*.

SET NOLANGUAGE

Disassociates the language associated with the indicated buffer.

Format

SET NOLANGUAGE

Qualifier
/BUFFER=buffer-name

Qualifier

/BUFFER=buffer-name
Indicates the buffer whose associated language you want to disassociate. The current buffer is the default.

Description

The SET NOLANGUAGE command disassociates the language currently in effect from the indicated buffer.

System buffers, such as $REVIEW, $MESSAGES, or $HELP have no languages associated with them; if you attempt to use this command with system buffers, you receive an error message.

Related Commands

DEFINE LANGUAGE
DELETE LANGUAGE
SET LANGUAGE
SHOW LANGUAGE

Example

```
LSE> SET NOLANGUAGE
```

Disassociates the currently associated language from the current buffer.

SET NOLIBRARY

Removes the specified SCA libraries from the current list of active libraries.

Format

SET NOLIBRARY *[library-spec[, . . .]]*

Qualifier	**Default**
/[NO]LOG	/LOG

Qualifier

/LOG (D)
/NOLOG
Indicates whether LSE reports removal of the libraries from the active list.

Parameter

library-spec[, . . .]
Specifies the libraries to be removed from the current active libraries list. If you omit this parameter, SCA removes all the active libraries from the list.

Description

The SET NOLIBRARY command allows you to selectively discard or purge specific SCA libraries from an active library list.

Related Commands

SET LIBRARY

Example

```
LSE>  SET NOLIBRARY PROJ:[USER.LIB1],PROJ:[USER.LIB2]
```

Removes the specified libraries from the current active libraries list.

See the chapter about using SCA libraries in the *VAX Language-Sensitive Editor and VAX Source Code Analyzer User Manual* for additional examples.

SET NOMODIFY

Sets a buffer to display-only (unmodifiable).

Format

SET NOMODIFY

Qualifier
/BUFFER=buffer-name

Qualifier

/BUFFER=buffer-name
Indicates the buffer that is to be set to display-only. The default is the current buffer.

Description

The SET NOMODIFY command sets a buffer to display-only (unmodifiable). After issuing this command, you cannot change the buffer's contents until you issue a SET MODIFY command.

Users of the DECwindows interface can cycle through Insert, Overstrike, and Nomodify by selecting the status line button and pressing MB1.

DECwindows Interface Equivalent

$$\textbf{Button}\text{: Buffer status line} \left\{ \begin{array}{l} \text{Insert} \\ \text{Overstrike} \\ \text{Nomodify} \end{array} \right\}$$

Related Commands

SET MODIFY
SHOW BUFFER [DECwindows interface]

Example

```
LSE>  SET NOMODIFY
```

Prevents you from modifying text that you had previously brought into the current buffer as modifiable.

SET NOOUTPUT_FILE

Disassociates the buffer from any output file.

Format

SET NOOUTPUT_FILE

Qualifier
/BUFFER=buffer-name

Qualifier

/BUFFER=buffer-name
Indicates the buffer whose output file is to be changed. The default is the current buffer.

Description

The SET NOOUTPUT_FILE command disassociates the indicated buffer from any output file. LSE uses output file associations when writing the buffer out to a file; thus, when you issue the SET NOOUTPUT_FILE command and then issue a COMPILE, EXIT, or WRITE command, you must supply LSE with a file name.

Related Commands

COMPILE
EXIT
SET OUTPUT_FILE
SHOW BUFFER [DECwindows interface]
WRITE

Example

```
LSE>  SET NOOUTPUT_FILE
```

Disassociates the current buffer from any output file. You must specify a file name to write the buffer to if you subsequently issue an EXIT or WRITE command.

SET NOOVERVIEW

Disables overview operations in the indicated buffer.

Format

SET NOOVERVIEW

Qualifier
/BUFFER=buffer-name

Qualifier

/BUFFER=buffer-name
Indicates the buffer in which overview operations are to be disabled. The default is the current buffer.

Description

The SET NOOVERVIEW command disables the use of overview operations in the indicated buffer. This disables the COLLAPSE, FOCUS, and VIEW SOURCE commands, and the use of the EXPAND command on an overview line.

Related Commands

COLLAPSE
EXPAND
FOCUS
SET OVERVIEW
VIEW SOURCE

SET NOSOURCE_DIRECTORY

Specifies a directory or directories to be removed from the list of source directories.

Format

SET NOSOURCE_DIRECTORY *[directory-spec [,directory-spec] . . .]*

Parameter

directory-spec [,directory-spec] . . .
Specifies a list of directory specifications to be removed from the list of source directories. If you do not specify any parameter, LSE removes all directories from the list of source directories.

Description

The SET NOSOURCE_DIRECTORY command removes the directories you specify from the list of source directories. If you do not specify any directories, LSE removes all directories in the source list from that list.

Related Commands

SET SOURCE_DIRECTORY

Examples

1.
```
LSE> SET NOSOURCE [PROJECT_DIRECTORY]
```

Removes the directory *PROJECT_DIRECTORY* from the list of source directories.

2.
```
LSE>  SET NOSOURCE/READ_ONLY [LIBRARY_DIRECTORY]
```

   Removes the directory *LIBRARY_DIRECTORY* from the set of read-only directories.

SET NOWRAP

Disables wrapping of the current line in the indicated buffer.

Format

SET NOWRAP

Qualifier
/BUFFER=buffer-name

Qualifier

/BUFFER=buffer-name
Indicates the buffer for which wrapping is to be disabled. The default is the current buffer.

Description

The SET NOWRAP command prevents the ENTER SPACE command (bound to the space bar by default) from performing a wrap operation on the current line in the indicated buffer.

Initially, wrapping of the current line is disabled.

Related Commands

ENTER LINE
ENTER SPACE
SET WRAP
SHOW BUFFER [DECwindows interface]

SET OUTPUT_FILE

Establishes the output file associated with the buffer.

Format

SET OUTPUT_FILE *file-spec*

Qualifier
/BUFFER=buffer-name

Qualifier

/BUFFER=buffer-name
Indicates the buffer whose output file is to be changed. The default is the current buffer.

Parameter

file-spec
Indicates the file specification for the output file.

Description

The SET OUTPUT_FILE command associates the specified output file with the indicated buffer. LSE uses output file associations when writing the buffer out to a file; this happens when you issue a COMPILE, EXIT, or WRITE command.

This command does not cause the buffer to be written to a file. You may also need to use the SET WRITE command.

Related Commands

COMPILE
EXIT
SET NOOUTPUT_FILE
SET WRITE
SHOW BUFFER [DECwindows interface]
WRITE

Example

```
LSE>  SET OUTPUT_FILE USER.TXT
```

Associates the output file *USER.TXT* with the current buffer. When you issue an EXIT or WRITE command, LSE writes the contents of that buffer to the file USER.TXT without prompting you for a file name.

SET OVERSTRIKE

Sets the text-entry mode of the indicated buffer to overstrike mode.

Format

SET OVERSTRIKE

Qualifier
/BUFFER=buffer-name

Qualifier

/BUFFER=buffer-name
Indicates the buffer whose text-entry mode is to be changed. The default is the current buffer.

Description

The SET OVERSTRIKE command sets the mode of the indicated buffer to overstrike mode. When you set this mode, typing a character replaces that character at the current cursor position. Pressing the Delete key replaces the character to the left of the cursor with a blank space.

The status line of each window displays the current text-entry mode for the associated buffer.

Users of the DECwindows interface can cycle through Insert, Overstrike, and Nomodify by selecting the status line button and pressing MB1.

DECwindows Interface Equivalent

$$\textbf{Button}\text{: Buffer status line} \left\{ \begin{array}{l} \text{Insert} \\ \text{Overstrike} \\ \text{Nomodify} \end{array} \right\}$$

Related Commands

CHANGE TEXT_ENTRY_MODE
SET INSERT

SET OVERVIEW

Enables overview operations in the indicated buffer.

Format

SET OVERVIEW

Qualifier
/BUFFER=buffer-name

Qualifier

/BUFFER=buffer-name
Indicates the buffer in which overview operations are to be enabled. The default is the current buffer.

Description

The SET OVERVIEW command enables the use of overview operations in the indicated buffer. This enables the COLLAPSE, FOCUS and VIEW SOURCE commands, as well as the use of the EXPAND command on an overview line.

By default, overview operations are allowed in a buffer when it is created. LSE disables overview operations in some system buffers that it creates. To see the current setting, use the SHOW BUFFER/FULL command.

Related Commands

COLLAPSE
EXPAND
FOCUS
SET NOOVERVIEW
VIEW SOURCE

SET READ_ONLY

Instructs LSE not to write the indicated buffer to a file when you exit from LSE or when you issue a COMPILE command.

Format

SET READ_ONLY

Qualifier
/BUFFER=buffer-name

Qualifier

/BUFFER=buffer-name
Indicates the buffer whose read_only/write state is to be changed. The default is the current buffer.

Description

The SET READ_ONLY command prevents LSE from writing the contents of the indicated buffer to a file when you exit from LSE or issue a COMPILE command. The LSE status line displays the read-only/write state.

Users of the DECwindows interface can switch between Write and Read-only by selecting the status line button and pressing MB1.

DECwindows Interface Equivalent

Button: Buffer status line $\left\{ \begin{array}{l} \text{Write} \\ \text{Read-only} \end{array} \right\}$

Related Commands

SET MODIFY
SET NOMODIFY
SET WRITE
SHOW BUFFER [DECwindows interface]

SET REVERSE

Sets the current direction of a buffer to reverse.

Format

SET REVERSE

Qualifier
/BUFFER=buffer-name

Qualifier

/BUFFER=buffer-name
Indicates the buffer whose direction is to be set to reverse. The default is the current buffer.

Description

The SET REVERSE command sets the current direction of the indicated buffer to reverse. The status line displays the current direction.

Users of the DECwindows interface can switch between Forward and Reverse by selecting the status line button and pressing MB1.

Keypad Equivalent

Key	Keypad Mode
KP5 REVERSE	EDT LK201, EDT VT100, EVE LK201
None	EVE VT100

DECwindows Interface Equivalent

Button: Buffer status line $\left\{ \begin{array}{l} \text{Forward} \\ \text{Reverse} \end{array} \right\}$

Related Commands

CHANGE DIRECTION
SET FORWARD

SET RIGHT_MARGIN

Sets the right margin for the indicated buffer.

Format

SET RIGHT_MARGIN *column-number*

Qualifier
/BUFFER=buffer-name

Qualifier

/BUFFER=buffer-name
Indicates the buffer whose right margin is to be changed. The default is the current buffer.

Parameter

column-number
Specifies the column for the right margin. The value must be an integer greater than the value set for the left margin.

Description

The SET RIGHT_MARGIN command sets the right margin of the indicated buffer to the column number you specify. By default, the right margin is set at column 80.

The right margin controls where LSE wraps words when you type new text. The FILL and ENTER SPACE commands also use this setting. To find out the setting of the right margin, use the SHOW BUFFER command.

Related Commands

SET LEFT_MARGIN
SHOW BUFFER [DECwindows interface]

Example

```
LSE>  SET RIGHT_MARGIN 65
```

Sets the right margin in the current buffer at column 65.

SET SCREEN

Sets specified characteristics of the screen.

Format

SET SCREEN *keyword-list*

Parameter

keyword-list
Indicates the screen characteristics to be set. The screen keywords are as follows:

BALANCE_WINDOWS (D)
NOBALANCE_WINDOWS
Specifies how LSE manages window length. If you specify BALANCE_WINDOWS, LSE adjusts all the window lengths on the screen to be, as nearly as possible, of equal lengths. This is the default value. If you specify NOBALANCE_WINDOWS, LSE splits the current window in half when it needs a new window, leaving all the other window lengths unchanged.

HEIGHT=n
Specifies the number of lines on the screen. The height, *n*, must be an integer in the range 11 through 62.

MAXIMUM_WINDOW_NUMBER=n
Specifies the maximum number of windows LSE creates when it displays information in a window as a result of issuing one of the following commands:

FIND
GOTO DECLARATION
GOTO SOURCE
INSPECT
REVIEW

LSE uses the MAXIMUM_WINDOW_NUMBER and MINIMUM_WINDOW_LENGTH settings to determine whether to add a window to the screen or reuse an existing window. LSE checks both settings and creates a new window only if both conditions are met.

The default value for MAXIMUM_WINDOW_NUMBER is 3. Specifying a value of 2 produces the two-window behavior previously associated with the commands listed under this keyword.

MINIMUM_WINDOW_LENGTH=n
Specifies a lower bound on the windows LSE creates. When you need to map a buffer to a window, LSE creates a new window as long as the window is not shorter than *n*.

LSE uses the MINIMUM_WINDOW_LENGTH and MAXIMUM_WINDOW_NUMBER settings to determine whether to add a window to the screen or reuse an existing window. LSE checks both settings and creates a new window only if both conditions are met.

WIDTH=n
Specifies the number of characters on each input or output line. The width, *n*, must be an integer in the range 1 through 252.

If you specify a width greater than 80, then LSE sets the terminal to 132-character mode. The initial setting is 80 characters.

WINDOW=n
Specifies the number of windows to display on the screen.

If you change the number of windows from one to two, LSE displays the current buffer in both windows. If you change the number of windows from two to one, LSE displays the current buffer in the single window. The initial setting is one window.

DECwindows Interface Equivalent

Pull-down menu: Customize --> Window Attributes . . .

Related Commands

SHOW SCREEN

Examples

1.
```
LSE>  SET SCREEN WIDTH=132
```
Sets your terminal to 132-character mode.

2.
```
LSE>  SET SCREEN MINIMUM_WINDOW_LENGTH=5,BALANCE_WINDOWS
```
For automatic window creation on a 24-line terminal, the keyword MINIMUM_WINDOW_LENGTH=5 allows up to four windows and the keyword BALANCE_WINDOWS causes the editor to keep all the windows approximately equal in length.

SET SCROLL_MARGINS

Delimits the lines at which the cursor triggers scrolling.

Format

SET SCROLL_MARGINS *top-line-count [%]*
bottom-line-count [%]

Parameters

top-line-count
Specifies the number of lines down from the top of a window at which you want downward scrolling to begin.

bottom-line-count
Specifies the number of lines up from the bottom of a window at which you want upward scrolling to begin.

%
Optionally specifies scroll margins as percentages of the window height, rounded to the nearest whole line count. This is useful when you have a workstation with screens of varying sizes.

Description

The SET SCROLL_MARGINS command specifies the lines at the top and bottom of the window at which scrolling is triggered by moving the cursor to these lines.

The scroll margins you set apply to all windows in the current editing session.

Examples

1. `LSE>  SET SCROLL_MARGINS 2 3`

   Sets the scroll margins at two lines from the top and three lines from the bottom of all windows in the current editing session.

2. `LSE>  SET SCROLL_MARGINS 10% 15%`

   Sets the scroll margins at 10% from the top and 15% from the bottom of all windows in the current editing session.

SET SEARCH

Sets text search options.

Format

SET SEARCH *keyword-list*

Parameter

keyword-list
Indicates the search mode settings. The keywords are as follows:

AUTO_REVERSE
NOAUTO_REVERSE

Specifies whether or not LSE searches in the current direction only, or searches in the opposite direction if the string is not found in the current direction. The initial setting is NOAUTO_REVERSE.

CASE_SENSITIVE
NOCASE_SENSITIVE

Specifies whether or not the SEARCH command matches case exactly or is insensitive to character case. The initial setting is NOCASE_SENSITIVE.

DIACRITICAL_SENSITIVE
NODIACRITICAL_SENSITIVE

Specifies whether or not the SEARCH command matches characters with diacritical markings exactly or is insensitive to diacritical markings. The initial setting is DIACRITICAL_SENSITIVE.

PATTERN=VMS
PATTERN=ULTRIX

Specifies either ULTRIX-style regular expressions or VMS-style patterns for the SEARCH/PATTERN command. The initial setting is VMS.

SPAN_SPACE
NOSPAN_SPACE

Determines whether LSE matches blanks in the search string exactly (NOSPAN_SPACE) or allows each blank to match sequences of one or more characters containing blanks and tabs and, at most, a single line break (SPAN_SPACE). The initial setting is NOSPAN_SPACE.

Description

The SET SEARCH command sets preconditions for matching text when you issue the SEARCH command.

DECwindows Interface Equivalent

Pull-down menu: Customize --> Search Attributes . . .

Related Commands

SEARCH
SHOW SEARCH

Example

```
LSE>  SET SEARCH CASE_SENSITIVE
```

Directs LSE to match case exactly when you issue a SEARCH command.

SET SELECT_MARK

Marks a position as one end of a selected range.

Format

SET SELECT_MARK

Description

The SET SELECT_MARK command marks a position as one end of a selected range. The selected range is the text between the select marker and the current cursor position; it is denoted by a reverse video display. This command is not valid if the select marker has already been set.

Keypad Equivalent

Key	Keypad Mode
Keypad period (.) SELECT	EDT LK201, EDT VT100, EVE LK201
None	EVE VT100

Related Commands

CANCEL SELECT_MARK
TOGGLE SELECT_MARK

SET SOURCE_DIRECTORY

Specifies a searchlist of directories to be used to find source files.

Format

SET SOURCE_DIRECTORY *directory-spec [,directory-spec] . . .*

Qualifiers	**Defaults**
/AFTER[=directory-spec]	/AFTER
/BEFORE[=directory-spec]	/AFTER

Qualifiers

/AFTER (D)
/AFTER[=directory-spec]
Specifies that LSE should insert the directory or directories indicated into the list of source directories in back of the directory you specify as the value on the qualifier. If you do not specify a *directory-spec* value, LSE adds the directory or directories to the end of the list.

If you do not specify either the /AFTER qualifier or the /BEFORE qualifier, LSE replaces the entire directory list.

/BEFORE
/BEFORE[=directory-spec]
Specifies that LSE should insert the directory or directories indicated into the list of source directories in front of the directory you specify as the value on the qualifier. If you do not specify a *directory-spec* value, the directory or directories are added at the front of the list.

If you do not specify either the /BEFORE qualifier or the /AFTER qualifier, LSE replaces the entire directory list.

Parameter

directory-spec [,directory-spec] . . .
Specifies one or more directory specifications. You can specify CMS$LIB as one directory specification; however, you may not get the results you expect if you set CMS$LIB as a source directory and do not issue the CMS command SET LIBRARY.

Description

The SET SOURCE_DIRECTORY command specifies the directories LSE uses to find source files when you issue the commands GOTO FILE, GOTO SOURCE, and READ.

The GOTO FILE and READ commands use this list of directories if you do not specify a directory for the file specified on the GOTO FILE or READ command.

The GOTO SOURCE command uses this list of directories if LSE does not find the source file specified in the SCA data file or the diagnostics file.

The logical name LSE$SOURCE stores the list of source directories.

Related Commands

SET CMS
SET NOSOURCE_DIRECTORY

Example

```
LSE>  SET SOURCE_DIRECTORY [],[MY_SOURCE_DIRECTORY],-
_LSE>  [PROJECT_SOURCE_DIRECTORY],CMS$LIB
```

Directs LSE to search for sources first in the current directory, then in the user's source directory, then in the project source directory, and finally in CMS$LIB.

SET TAB_INCREMENT

Specifies logical tab stops in the indicated buffer.

Format

SET TAB_INCREMENT *number*

Qualifier
/BUFFER=buffer-name

Qualifier

/BUFFER=buffer-name
Indicates the buffer whose tab increment is to be changed. The default is the current buffer.

Parameter

number
Specifies the interval for setting tab stops.

Description

The SET TAB_INCREMENT command specifies the number of columns between the tab stops for the indicated buffer. Tab stops are set beginning with column 1. All previous tab stops are cleared.

Related Commands

ENTER TAB
SHOW BUFFER [DECwindows interface]
TAB

Example

```
LSE>  SET TAB_INCREMENT 4
```

Sets tab stops in columns 1, 5, 9, 13, and so on.

SET WRAP

Enables wrapping in the indicated buffer. LSE automatically splits the current line at the right-margin setting when you type text past the right margin.

Format

SET WRAP

Qualifier
/BUFFER=buffer-name

Qualifier

/BUFFER=buffer-name
Indicates the buffer for which wrapping is to be enabled. The default is the current buffer.

Description

The SET WRAP command enables the ENTER SPACE and ENTER LINE commands to perform a wrap operation in the indicated buffer.

Initially, wrapping is disabled.

Related Commands

ENTER LINE
ENTER SPACE
SET NOWRAP
SHOW BUFFER [DECwindows interface]

SET WRITE

Instructs LSE to write the contents of the indicated buffer to a file when you exit from LSE or issue a COMPILE command.

Format

SET WRITE

Qualifier
/BUFFER=buffer-name

Qualifier

/BUFFER=buffer-name
Indicates the buffer whose read-only/write state is to be changed. The default is the current buffer.

Description

The SET WRITE command reverses the action of the SET READ_ONLY command. When you exit from LSE or issue a COMPILE command, LSE writes the contents of the indicated buffer to a file. The status line displays the setting of the read-only/write state.

If the indicated buffer is unmodifiable, issuing a SET WRITE command is also equivalent to issuing a SET MODIFY command. If the directory for the file associated with the buffer is read-only, LSE displays a message informing you of that fact.

Users of the DECwindows interface can switch between Write and Read-only by selecting the status line button and pressing MB1.

DECwindows Interface Equivalent

$$\textbf{Button}\text{: Buffer status line} \left\{ \begin{array}{l} \text{Write} \\ \text{Read-only} \end{array} \right\}$$

Related Commands

SET OUTPUT_FILE
SET READ_ONLY
SHOW BUFFER [DECwindows interface]

SHIFT

Shifts the window horizontally to the left or right one column.

Format

SHIFT

Qualifiers	Defaults
/CURRENT	/CURRENT
/FORWARD	/CURRENT
/REVERSE	/CURRENT

Qualifiers

/CURRENT (D)
Specifies the current direction for the shift.

/FORWARD
Shifts the window to the right so that you can view formerly hidden text to the right of the original text.

/REVERSE
Shifts the window to the left so that you can view any text hidden by a SHIFT/FORWARD command.

Description

The SHIFT command shifts or moves the display window horizontally to the left or right one column. The SHIFT qualifiers refer to the direction of window movement with respect to the text. When used with a repeat count, the value of the repeat count determines the extent of the shift (see the REPEAT command).

Users of the DECwindows interface can achieve similar results by using the horizontal scroll bar.

Example

```
LSE> REPEAT 3 SHIFT/FORWARD
```

Moves the display window three columns to the right.

SHOW ADJUSTMENT

Displays the characteristics of indicated adjustments.

Format

SHOW ADJUSTMENT *[adjustment-name]*

Qualifier
/LANGUAGE=language-name

Qualifier

/LANGUAGE=language-name
Associates a language with the specified adjustments. If you do not specify a language, LSE displays information about adjustments associated with the correct language. If you specify /LANGUAGE=*, LSE displays information on any adjustment that matches the adjustment name, regardless of the language for which it is defined.

Parameter

adjustment-name
Specifies which adjustments are to be shown. If you omit this parameter, LSE assumes you have specified a wildcard adjustment name.

Description

The SHOW ADJUSTMENT command displays the definitions and characteristics of an adjustment or adjustments.

Related Commands

DEFINE ADJUSTMENT
DELETE ADJUSTMENT
EXTRACT ADJUSTMENT

Example

```
LSE>  SHOW ADJUSTMENT then
```

Displays all the characteristics defined for the adjustment *then*.

SHOW ALIAS

Displays information on the specified alias.

Format

SHOW ALIAS *[alias-name]*

Qualifiers	**Default**
/BRIEF	See text
/FULL	See text
/LANGUAGE=language-name	

Qualifiers

/BRIEF
Causes LSE to display (in tabular format) the alias name and equivalent string.

If you specify a wildcard expression for the parameter or if LSE assumes one, /BRIEF is the default.

/FULL
Causes LSE to display the alias name and equivalent string in list format.

If you specify an explicit name for the parameter, /FULL is the default.

/LANGUAGE=language-name
Specifies the language associated with the alias. The default is the current language.

Parameter

alias-name
Specifies the name of the alias whose characteristics are to be displayed. If this parameter is omitted, a wildcard alias name is assumed.

SHOW ALIAS

Description

The SHOW ALIAS command displays information on an alias you defined using the DEFINE ALIAS command.

Related Commands

DEFINE ALIAS

Example

```
LSE>  SHOW ALIAS
```

Displays one line of information for each of the aliases you have currently defined.

SHOW BUFFER

Displays the characteristics of one or more buffers.

Format

SHOW BUFFER *[buffer-name]*

Qualifiers	**Defaults**
/ALL_BUFFERS	/USER_BUFFERS
/BRIEF	See text
/FULL	See text
/SYSTEM_BUFFERS	/USER_BUFFERS
/USER_BUFFERS	/USER_BUFFERS

Qualifiers

/ALL_BUFFERS
Specifies all buffers to be displayed when a wildcard buffer name is specified or assumed. LSE ignores this qualifier if you specify an explicit buffer name.

/BRIEF
Causes the current window to display (in tabular format) the name, number of text lines, and information about whether the buffer is modified, compiled, reviewed, or modifiable.

If you move the cursor to a line containing a buffer name and press the Select key, LSE performs a GOTO BUFFER command for that buffer. If you move the cursor to a line containing a buffer name and press the Remove key, LSE performs a DELETE BUFFER command for that buffer. In DECwindows mode, you can perform a GOTO BUFFER for a buffer displayed in the list by pressing MB1 twice on the line containing the buffer name.

If you specify a wildcard expression, or if LSE assumes one, /BRIEF is the default.

/FULL
Causes LSE to list all the information available about each indicated buffer, including associated input and output files, language, and all the buffer attributes that you can set, such as margins and text-entry mode.

If you specify an explicit buffer, /FULL is the default.

/SYSTEM_BUFFERS
Specifies that only system buffers be displayed when a wildcard buffer name is specified or assumed. LSE ignores this qualifier if you specify an explicit buffer name.

/USER_BUFFERS (D)
Specifies that only user buffers be displayed when a wildcard buffer name is specified or assumed. LSE ignores this qualifier if you specify an explicit buffer name.

Parameter

buffer-name
Specifies the name of the buffer or buffers whose characteristics are to be displayed. If you specify a null buffer name (""), the current buffer is assumed. If this parameter is omitted, a wildcard buffer name is assumed.

Description

The SHOW BUFFER command displays information about the indicated buffer or buffers.

DECwindows Interface Equivalent

SHOW BUFFER
Pull-down menu: Display --> Show Buffers

Related Commands

NEXT BUFFER
PREVIOUS BUFFER
SET AUTO_ERASE
SET LEFT_MARGIN
SET NOAUTO_ERASE
SET NOOUTPUT_FILE
SET NOWRAP
SET OUTPUT_FILE
SET READ_ONLY
SET RIGHT_MARGIN
SET TAB_INCREMENT
SET WRAP
SET WRITE

Example

```
LSE> SHOW BUFFER/SYSTEM
```

Displays the name, number of text lines, and status (read-only, modifiable) for each system buffer.

SHOW CMS

Displays the current CMS settings, which are the initial settings unless you have changed them using the SET CMS command.

Format

SHOW CMS

Description

The SHOW CMS command lists all the CMS settings specified by the qualifiers to the SET CMS command. If you have not issued a SET CMS command, the listed CMS settings reflect initial conditions.

For users of the DECwindows interface, the SHOW CMS command displays a CMS Attribute dialog box to let you change the current CMS settings.

DECwindows Interface Equivalent

Pull-down menu: Customize --> CMS Attributes . . .

Related Commands

SET CMS

SHOW COMMAND

Displays the characteristics of a user-defined command.

Format

SHOW COMMAND *[command-name]*

Parameter

command-name
Specifies the name of the command whose characteristics are to be displayed. If you omit this parameter, LSE displays information on all user-defined commands.

Description

The SHOW COMMAND command displays the characteristics of a command you have defined using the DEFINE COMMAND command.

Related Commands

DEFINE COMMAND

Example

```
LSE>  SHOW COMMAND
```

Displays the command definition for each current user-defined command.

SHOW DEFAULT_DIRECTORY

Displays the current default device and directory.

Format

SHOW DEFAULT_DIRECTORY

Description

The SHOW DEFAULT_DIRECTORY command displays the current device and directory names, along with any equivalence strings. You can change the default with the LSE command SET DEFAULT_DIRECTORY.

Related Commands

SET DEFAULT_DIRECTORY

Example

```
LSE>  SHOW DEFAULT_DIRECTORY
```

Displays the current device and directory names.

SHOW DIRECTORY

Displays the setting of the SET DIRECTORY command.

Format

SHOW DIRECTORY

Description

The SHOW DIRECTORY command displays the list of directories specified by the SET DIRECTORY command.

Related Commands

SET DIRECTORY

SHOW KEY

Displays the definitions bound to the normal state and GOLD state of any defined key.

Format

SHOW KEY *key-specifier*

Qualifiers	**Defaults**
/BRIEF	/BRIEF
/FULL	/BRIEF

Qualifiers

/BRIEF (D)
Indicates how much information you want displayed. The /BRIEF qualifier instructs LSE to display only key names and the commands associated with them.

/FULL
Indicates how much information you want displayed. The /FULL qualifier instructs LSE to display topics, legends, and remarks as well as the key names and commands.

Parameter

key-specifier
Specifies the name of the key whose definitions are to be displayed. You may use a wildcard character on the command line to specify all defined keys or a group of related keys. If you press the Return key before specifying a key, LSE supplies quotation marks to any specifier you type at the prompt; thus, LSE interprets an asterisk specified at the prompt as the asterisk key on the keyboard and not as a wildcard character.

To specify key combinations beginning with the PF1 key, use the prefix GOLD/. To specify combinations by using the control key, use the form CTRL/*x*, where *x* may be the letters A through Z.

Description

The SHOW KEY command displays the definitions bound to the normal state and the GOLD state of any or all keyboard keys. This includes both the default bindings and those keys you have bound using the DEFINE KEY command.

The SHOW KEY command accepts key names that are valid for the DEFINE KEY command if you have used the following syntax for the key being defined:

```
LSE> DEFINE KEY "CTRL/A" "SHOW BUFFER"
```

Related Commands

DEFINE KEY
DELETE KEY

Examples

1. `LSE> SHOW KEY PF2`

   Displays the definitions currently bound to the *PF2* key.

2. `LSE> SHOW KEY CTRL*`

   Displays the definitions currently bound to all key sequences that begin with *CTRL*.

SHOW KEYWORDS

Displays the characteristics of the specified keyword list.

Format

SHOW KEYWORDS *[keyword-list-name]*

Qualifiers	**Defaults**
/BRIEF	/BRIEF
/FULL	/BRIEF

Qualifiers

/BRIEF (D)
Indicates how much information you want displayed. The /BRIEF qualifier causes LSE to display (in tabular format) the name of the indicated keyword list.

/FULL
Indicates how much information you want displayed. The /FULL qualifier causes LSE to display all the information available about the indicated keyword list, as specified by the current DEFINE KEYWORDS command (see the list of qualifiers for the DEFINE KEYWORDS command).

Parameter

keyword-list-name
Specifies the keyword list or keyword lists about which information is wanted. By default, LSE displays information about the keyword list associated with the current buffer.

Description

The SHOW KEYWORDS command displays the characteristics of a specified keyword list. The keyword list must be known to LSE.

Related Commands

DEFINE KEYWORDS
DELETE KEYWORDS
EXTRACT KEYWORDS

Example

```
LSE>  SHOW KEYWORDS author_name
```

Displays the characteristics associated with the specified keyword list.

SHOW LANGUAGE

Displays the characteristics of the specified language.

Format

SHOW LANGUAGE *[language-name]*

Qualifiers	**Defaults**
/BRIEF	/FULL
/FULL	/FULL

Qualifiers

/BRIEF
Indicates how much information you want displayed. The /BRIEF qualifier causes LSE to display (in tabular format) the name and file type of the indicated language.

/FULL (D)
Indicates how much information you want displayed. The /FULL qualifier causes LSE to display all the information available about the indicated language, as specified by the current DEFINE LANGUAGE command (see the list of qualifiers for the DEFINE LANGUAGE command).

Parameter

language-name
Specifies the language or languages about which information is wanted. By default, LSE displays information about the language associated with the current buffer.

Description

The SHOW LANGUAGE command displays the characteristics of a specified language. The language must be known to LSE.

Related Commands

DEFINE LANGUAGE
MODIFY LANGUAGE
SET LANGUAGE

Example

```
LSE> SHOW LANGUAGE Pascal
```

Displays compiler, file type, punctuation, and other characteristics associated with the programming language *Pascal.*

SHOW LIBRARY

Displays the directory specification for all active SCA libraries.

Format

SHOW LIBRARY

Qualifiers	**Defaults**
/BRIEF	/BRIEF
/FULL	/BRIEF

Qualifiers

/BRIEF (D)
Displays only the directory specification for all active libraries.

/FULL
Displays all information about all active SCA libraries.

Description

The SHOW LIBRARY command displays the directory specifications for all active SCA libraries.

Related Commands

CREATE LIBRARY
SET LIBRARY
SET NOLIBRARY

Example

```
$ SCA SHOW LIBRARY
```

Displays the location of the current library.

SHOW MARK

Displays the setting of the specified mark.

Format

SHOW MARK *[marker-name]*

Qualifiers	**Defaults**
/BRIEF	See text
/FULL	See text

Qualifiers

/BRIEF
Indicates how much information you want displayed. The /BRIEF qualifier causes LSE to display (in tabular format) the name and associated buffer for each marker currently set.

If you specify a wildcard expression for the parameter, or if LSE assumes one, /BRIEF is the default.

/FULL
Indicates how much information you want displayed. The /FULL qualifier causes LSE to list all the information available about each indicated marker, including the associated text.

If you specify an explicit marker for the parameter, /FULL is the default.

Parameter

marker-name
Specifies the name of the marker whose characteristics are to be displayed. If you omit this parameter, LSE displays the names of all the markers you have set.

Description

The SHOW MARK command displays the name or names of markers associated with the current buffer.

Related Commands

SET MARK

Example

```
LSE>  SHOW MARK
```

Lists the currently set marker names and their associated buffers.

SHOW MODE

Displays the current settings for modes set with the SET MODE command.

Format

SHOW MODE

Description

The SHOW MODE command displays the current mode settings for key words used with the SET MODE command.

For users of the DECwindows interface, the SHOW MODE command uses the Global Attribute dialog box to display the modes. This dialog box permits you to change the mode settings.

DECwindows Interface Equivalent

Pull-down menu: Customize --> Global Attributes . . .

Related Commands

SET MODE

Example

```
LSE>  SHOW MODE
```

Displays currently set editing-mode status for warning bells, keypad mode, and menu-display characters.

SHOW MODULE

Displays information about SCA library modules.

Format

SHOW MODULE *[module-name[, . . .]]*

Qualifiers	**Defaults**
/ALL	/VISIBLE
/BRIEF	See text
/FULL	See text
/HIDDEN	/VISIBLE
/LIBRARY=library-spec	/LIBRARY=*
/OUTPUT[=file-spec]	
/VISIBLE	/VISIBLE

Qualifiers

/ALL
Specifies that SCA display both hidden and visible modules.

/BRIEF
Indicates how much information you want displayed. The /BRIEF qualifier causes SCA to display selected information about each indicated module in tabular format. For an example, see the chapter about getting started with SCA in the *VAX Language-Sensitive Editor and VAX Source Code Analyzer User Manual.*

If you specify a wildcard expression for the parameter, or if SCA assumes one, /BRIEF is the default.

/FULL
Indicates how much information you want displayed. The /FULL qualifier causes SCA to list all information available about each indicated module. For an example, see the chapter about getting started with SCA in the *VAX Language-Sensitive Editor and VAX Source Code Analyzer User Manual.*

If you specify an explicit name for the parameter, /FULL is the default.

/HIDDEN
Specifies that SCA display only hidden modules.

/LIBRARY=library-spec
/LIBRARY=* (D)
Specifies an SCA library containing the module to be displayed. The library must be one of the current SCA libraries established by a SET LIBRARY command.

If you do not specify the /LIBRARY qualifier, SCA assumes you have specified all current SCA libraries.

/OUTPUT[=file-spec]
Directs command output to a file rather than to the $SHOW buffer. The default output file specification is SCA.LIS.

/VISIBLE (D)
Specifies that SCA display only visible modules.

Parameter

module-name[, . . .]
Specifies the modules to be displayed. If you omit this parameter, SCA displays all modules. You may specify wildcard module names.

Description

The SHOW MODULE command displays information about modules in SCA libraries.

Related Commands

SET LIBRARY

Example

```
$ SCA SHOW MODULE
```

Displays all of the source module information from the library in an abbreviated (/BRIEF by default) form.

SHOW PACKAGE

Displays the characteristics of indicated packages.

Format

SHOW PACKAGE *package-name*

Qualifiers	**Defaults**
/BRIEF	See text
/FULL	See text

Qualifiers

/BRIEF
Indicates how much information you want displayed. The /BRIEF qualifier causes LSE to display (in tabular format) the name of each indicated package.

If you specify a wildcard expression for the parameter, or if LSE assumes one, /BRIEF is the default.

/FULL
Indicates how much information you want displayed. The /FULL qualifier causes LSE to display all the information available about each indicated package, as specified by the current DEFINE PACKAGE command (see the list of qualifiers for the DEFINE PACKAGE command).

If you specify an explicit name for the parameter, /FULL is the default.

Parameter

package-name
Specifies the name of the package to be displayed. You may use wildcards. If you omit this parameter, LSE displays the status of all known packages.

Description

The SHOW PACKAGE command displays the status of the indicated package or packages. By default, LSE gives a brief description.

Related Commands

DEFINE PACKAGE

Example

```
LSE> SHOW PACKAGE system_services
```

Displays all the characteristics defined for the package *system_services*.

SHOW PARAMETER

Displays the characteristics of indicated parameters.

Format

SHOW PARAMETER *[parameter-name]*

Qualifiers	**Defaults**
/BRIEF	See text
/FULL	See text
/LANGUAGE=language-name	
/PACKAGE=package-name	

Qualifiers

/BRIEF
Indicates how much information you want displayed. The /BRIEF qualifier causes LSE to display (in tabular format) the name and package associated with each indicated parameter. If you specify a wildcard expression for the parameter, or if LSE assumes one, /BRIEF is the default.

/FULL
Indicates how much information you want displayed. The /FULL qualifier causes LSE to display all the information available about each indicated parameter, as specified by the current DEFINE PARAMETER command (see the list of qualifiers for the DEFINE PARAMETER command).

If you specify an explicit name for the parameter, /FULL is the default.

/LANGUAGE=language-name
Shows only those parameters associated with the indicated language. If you do not specify a language, LSE uses the current language. If you specify /LANGUAGE=*, LSE displays information on any parameter that matches the parameter name, regardless of the language for which it is defined.

The /LANGUAGE qualifier is mutually exclusive with the /PACKAGE qualifier.

/PACKAGE=package-name
Specifies the name of the package with which the parameter is associated. The /PACKAGE qualifier is mutually exclusive with the /LANGUAGE qualifier.

Parameter

parameter-name
Specifies which parameters are to be shown. If you omit this name, LSE assumes you have specified a wildcard parameter name.

Description

The SHOW PARAMETER command displays the definitions and characteristics of one or more parameters.

Related Commands

DEFINE PARAMETER

Example

```
LSE> SHOW PARAMETER id
```

Displays all the characteristics defined for the parameter *id*.

SHOW PLACEHOLDER

Displays the characteristics of indicated placeholders.

Format

SHOW PLACEHOLDER *[placeholder-name]*

Qualifiers	**Defaults**
/BRIEF	See text
/FULL	See text
/LANGUAGE=language-name	

Qualifiers

/BRIEF
Indicates how much information you want displayed. The /BRIEF qualifier causes LSE to display (in tabular format) the name and description of each placeholder.

If you specify a wildcard expression for the parameter, or if LSE assumes one, /BRIEF is the default.

/FULL
The /FULL qualifier causes LSE to display all the information available about each indicated placeholder, as specified by the current DEFINE PLACEHOLDER command (see the list of qualifiers for the DEFINE PLACEHOLDER command).

If you specify an explicit name for the parameter, /FULL is the default.

/LANGUAGE=language-name
Associates a language with the specified placeholders. If you do not specify a language, LSE associates placeholders with the current language. If you specify /LANGUAGE=*, LSE displays information on any placeholder that matches the placeholder name, regardless of the language for which it is defined.

Parameter

placeholder-name
Specifies which placeholders are to be shown. If you omit this parameter, LSE assumes you have specified a wildcard placeholder name.

Description

The SHOW PLACEHOLDER command displays the definitions and characteristics of a placeholder or placeholders.

Related Commands

DEFINE PLACEHOLDER

Example

```
LSE> SHOW PLACEHOLDER parameter
```

Displays all the characteristics defined for the placeholder *parameter*.

SHOW QUERY

Displays information about one or more current SCA query sessions.

Format

SHOW QUERY *[query-name,...]*

Qualifiers	**Defaults**
/BRIEF	/BRIEF
/FULL	/BRIEF

Qualifiers

/BRIEF (D)
Indicates how much information you want to be displayed. The /BRIEF qualifier causes SCA to display (in tabular format) the query name, query expression, and description for the indicated query.

/FULL
Indicates how much information you want to be displayed. The /FULL qualifier causes SCA to display all information about the indicated query.

Parameter

query-name
Specifies the name of the query to be displayed. If you specify a null query name (""), SCA assumes you mean the current query. If you omit this parameter, SCA assumes you have specified an asterisk (*).

Related Commands

FIND
GOTO QUERY

Example

```
LSE> SHOW QUERY
```

Displays one line of information on all current SCA queries.

SHOW ROUTINE

Displays the characteristics of one or more routines.

Format

SHOW ROUTINE *[routine-name]*

Qualifiers	**Defaults**
/BRIEF	See text
/FULL	See text
/LANGUAGE=language-name	
/PACKAGE=package-name	

Qualifiers

/BRIEF
Indicates how much information you want displayed. The /BRIEF qualifier causes LSE to display (in tabular format) the name and package associated with each routine.

If you specify a wildcard expression for the parameter, or if LSE assumes one, /BRIEF is the default.

/FULL
Indicates how much information you want displayed. The /FULL qualifier causes LSE to display all the information available about each indicated routine, as specified by the current DEFINE ROUTINE command (see the list of qualifiers for the DEFINE ROUTINE command).

If you specify an explicit name for the parameter, /FULL is the default.

/LANGUAGE=language-name
Shows routines that are associated with the indicated language. If you do not specify a language, LSE uses the current language. If you specify /LANGUAGE=*, LSE displays information on any routine that matches the routine name, regardless of the language for which it is defined. The /LANGUAGE and /PACKAGE qualifiers are mutually exclusive.

/PACKAGE=package-name
Specifies the name of the package with which the routine is associated. The /PACKAGE and /LANGUAGE qualifiers are mutually exclusive.

Parameter

routine-name
Indicates which routines are to be displayed. If you omit this parameter, LSE assumes you have specified a wildcard routine name.

Description

The SHOW ROUTINE command displays the definitions and characteristics of one or more routines.

Related Commands

DEFINE ROUTINE

Example

```
LSE>  SHOW ROUTINE sys$add_holder
```

Displays all the characteristics defined for the routine *sys$add_holder*.

SHOW SCREEN

Displays the current values set with the SET SCREEN command.

Format

SHOW SCREEN

Description

The SHOW SCREEN command displays the current values for keywords used with the SET SCREEN command.

For users of the DECwindows interface, the SHOW SCREEN command uses the Window Attributes dialog box to display the screen attributes. This dialog box permits you to change the screen settings.

DECwindows Interface Equivalent

Pull-down menu: Customize --> Window Attributes . . .

Related Commands

SET SCREEN

Example

```
LSE>  SHOW SCREEN
```

Displays all the screen attributes set by the WIDTH, HEIGHT, WINDOW, BALANCE_WINDOWS, and MINIMUM_WINDOW_LENGTH keywords of the SET SCREEN command, and the fonts set by the SET FONT command.

SHOW SEARCH

Displays the settings of text-search options.

Format

SHOW SEARCH

Description

The SHOW SEARCH command shows the current settings of the various text-search options. In DECwindows mode, LSE uses the Search Attributes dialog box to display the search settings. This dialog box permits you to change the settings.

DECwindows Interface Equivalent

Pull-down menu: Customize --> Search Attributes . . .

Related Commands

SEARCH
SET SEARCH

SHOW SOURCE_DIRECTORY

Displays the setting of the SET SOURCE_DIRECTORY command.

Format

SHOW SOURCE_DIRECTORY

Description

The SHOW SOURCE_DIRECTORY command displays the list of directories specified by the SET SOURCE_DIRECTORY command.

Related Commands

SET SOURCE_DIRECTORY

SHOW SUMMARY

Shows statistics and other information about LSE.

Format

SHOW SUMMARY

Description

The SHOW SUMMARY command shows statistics and other information about LSE as follows:

- Version number of the software
- Current journal file specification (if any)
- Current section file specification
- Total number of buffers (system- and user-created)
- Modules used in the section file
- Other information about the LSE configuration

This information is useful for VAXTPU programming, or in case you need to submit a software performance report (SPR).

To scroll through the list, use the Next Screen key and the Prev Screen key. To return to the buffer you were editing, press the Return key.

SHOW TAG

Displays the characteristics of indicated tags.

Format

SHOW TAG *[tag-name]*

Qualifiers	Defaults
/BRIEF	See text
/FULL	See text
/LANGUAGE=language-name	

Qualifiers

/BRIEF
Indicates how much information you want displayed. The /BRIEF qualifier causes LSE to display (in tabular format) the name and description of each tag.

If you specify a wildcard expression for the parameter, or if LSE assumes one, /BRIEF is the default.

/FULL
The /FULL qualifier causes LSE to display all the information available about each indicated tag, as specified by the current DEFINE TAG command (see the list of qualifiers for the DEFINE TAG command).

If you specify an explicit name for the parameter, /FULL is the default.

/LANGUAGE=language-name
Associates a language with the specified tags. If you do not specify a language, LSE associates tags with the current language. If you specify /LANGUAGE=*, LSE displays information on any tag that matches the tag name, regardless of the language for which it is defined.

Parameter

tag-name
Specifies which tags are to be shown. If you omit this parameter, LSE assumes you have specified a wildcard tag name.

Description

The SHOW TAG command displays the definitions and characteristics of a tag or tags.

Related Commands

DEFINE TAG
DELETE TAG
EXTRACT TAG

Example

```
LSE>  SHOW TAG parameter
```

Displays all the characteristics defined for the tag *parameter*.

SHOW TOKEN

Displays the characteristics of one or more tokens.

Format

SHOW TOKEN *[token-name]*

Qualifiers	**Defaults**
/BRIEF	See text
/FULL	See text
/LANGUAGE=language-name	

Qualifiers

/BRIEF
Indicates how much information you want displayed. The /BRIEF qualifier causes LSE to display (in tabular format) the name and description of each token.

If you specify a wildcard expression for the parameter, or if LSE assumes one, /BRIEF is the default.

/FULL
Indicates how much information you want displayed. The /FULL qualifier causes LSE to display all the information available about each indicated token, as specified by the current DEFINE TOKEN command (see the list of qualifiers for the DEFINE TOKEN command).

If you specify an explicit name for the parameter, /FULL is the default.

/LANGUAGE=language-name
Associates a language with the specified tokens. If you do not specify a language, LSE associates tokens with the current language. If you specify /LANGUAGE=*, LSE displays any tokens that match the token name, regardless of the language for which it is defined.

Parameter

token-name
Indicates which tokens are to be displayed. If you omit this parameter, LSE assumes you have specified a wildcard token name.

Description

The SHOW TOKEN command displays the definitions and characteristics of one or more tokens.

Related Commands

DEFINE TOKEN

Example

```
LSE> SHOW TOKEN/LANGUAGE=EXAMPLE ASSIGNMENT
```

Displays the characteristics defined for the token *ASSIGNMENT* associated with the language *EXAMPLE*.

SHOW VERSION

Displays the current version of LSE and SCA.

Format

SHOW VERSION

Description

The SHOW VERSION command displays the current version of LSE and SCA.

If you are using SCA alone, only the SCA version is displayed. If you are using LSE, the LSE version is displayed, and the SCA version is displayed if SCA is installed on your system.

Examples

1.
```
$SCA SHOW VERSION
%SCA-S-VERSION, this is SCA version V2.0
$
```

Displays the version of SCA that you are using.

2.
```
LSE> SHOW VERSION
This is LSE version V3.0
This is SCA version V2.0
LSE>
```

Displays the version of LSE and SCA that you are using.

SHRINK WINDOW

Shrinks the current window.

Format

SHRINK WINDOW *line-count*

Parameter

line-count
Indicates the number of screen lines you want to subtract from the current window. The maximum size of a window depends on the size and type of the terminal screen you are using. The minimum size is one line of text and one line for the status line.

Description

The SHRINK WINDOW command shrinks the window that the text cursor is in (if you are using more than one window). LSE enlarges the other window (or windows) accordingly.

Related Commands

ENLARGE WINDOW

Example

```
LSE> SHRINK WINDOW 5
```

Subtracts five lines from the current window and apportions the lines to the other window (or windows) you have on the screen.

SPAWN

Spawns a subprocess running the DCL command interpreter and suspends the editing session.

NOTE

This function is not available in DECwindows; any attempt to invoke it incurs an error.

Format

SPAWN *[command]*

Parameter

command
Specifies a command line to be executed by the spawned subprocess. If you specify this parameter, the subprocess ends and LSE regains control upon completion of the command.

Description

The SPAWN command suspends the current LSE session and connects your terminal to a new VMS process at the DCL level. To resume your editing session, log out of the VMS process, or use the DCL ATTACH command to resume the editor process.

This command is useful for running screen-oriented programs and VMS utilities without ending the current editing session.

Related Commands

ATTACH

Example

```
LSE> SPAWN
```

Connects you to a new subprocess. The DCL dollar sign ($) prompt signifies subprocess connection.

SPELL

Runs DECspell to check the currently selected text or the entire buffer.

Format

SPELL

Description

The SPELL command runs DECspell (if it is installed on your system) to check the currently selected text or the entire buffer.

Use the following steps:

1. Select the text you want to check. If you do not select any text, SPELL checks the entire buffer.
2. Issue the SPELL command. If you select less than a full line, LSE extends the selected range to include the beginning and end of the line containing the range.

If the selected range (or the entire buffer if you do not select any text) contains any overview records, a message informs you that the operation cannot be performed.

LSE spawns a subprocess to run DECspell and writes out the current buffer or selected range to a temporary file in SYS$SCRATCH. (The name of the temporary file uses the subprocess PID.)

When SPELL finishes, LSE replaces the buffer or selected range with the new version of the temporary file (with corrections) and deletes any old versions of the temporary file. You then resume editing.

Do not use CTRL/Y with SPELL. CTRL/Y deletes lines in the temporary output file, and therefore destroys the selected range or current buffer.

You use the SPELL command only with DECwindows.

SPLIT WINDOW

Divides the current window into two or more windows.

Format

SPLIT WINDOW *[window-count]*

Parameter

window-count
Specifies the number of windows to create. The maximum size of a window depends on the size and type of the terminal screen you are using. The minimum size is one line of text and one line for the status line.

The text cursor appears in the lowest of the new windows.

Description

The SPLIT WINDOW command splits the current window into two or more windows. LSE displays the current buffer in each of the new windows.

DECwindows Interface Equivalent

SPLIT WINDOW 2
Pull-down menu: Display --> Split Window

Related Commands

CHANGE WINDOW_MODE
DELETE WINDOW

Example

```
LSE> SPLIT WINDOW 4
```

Splits the current window into four windows with the current buffer displayed in each.

SUBSTITUTE

Replaces occurrences of one text string with another.

Format

SUBSTITUTE *search-string*
replace-string

Qualifiers	**Defaults**
/ALL	/CONFIRM
/[NO]CASE_MATCHING	/NOCASE_MATCHING
/CONFIRM	/CONFIRM
/DIALOG	/NODIALOG
/[NO]PATTERN	/NOPATTERN
/SINGLE	/CONFIRM

Qualifiers

/ALL
Specifies that all occurrences of the search string are to be replaced with the replace string. Specifying the /ALL qualifier causes LSE to perform all the indicated substitutions without prompting you for further instructions.

/CASE_MATCHING
/NOCASE_MATCHING (D)
Specifies whether LSE uses the case of words in the search string to determine the case for the replacement string. The four conditions are: uppercase, lowercase, capitalized, or undetermined. For example, if a word in the search string is all uppercase, all the letters of the corresponding word in the replacement string become uppercase. If a word in the search string does not match the criteria for uppercase, lowercase, or capitalization, or there are no alphabetic characters in the search string word, its case is undetermined and LSE does not modify the case of the corresponding word in the replacement string.

If the replacement string contains more than one word, LSE respectively matches the case of words in the replacement string with the case of the corresponding words in the search string. If the search string contains fewer words than the replacement string, LSE matches the case of the additional words of the replacement string with the case of the last word in the search string.

Specifying the /NOCASE_MATCHING qualifier causes LSE not to modify the case of the replacement string to match that of the search string.

/CONFIRM (D)
Instructs LSE to prompt you for a confirmation at each occurrence before performing a substitution. If you specify the /CONFIRM qualifier, LSE highlights each occurrence of the search string located by the search and prompts you for an action.

- YES instructs LSE to replace this occurrence.
- NO instructs LSE not to replace this occurrence but to proceed with the command.
- QUIT ends the command without replacing this occurrence and stops the SUBSTITUTE operation.
- ALL replaces this occurrence and all remaining occurrences without further prompting.

/DIALOG
/NODIALOG (D)
Instructs LSE to use a dialog box to prompt you for parameters and qualifier values. If you specify this qualifier, the command parameters are optional. If you supply command parameters and qualifiers with the /DIALOG qualifier, LSE uses those parameters and qualifiers to set the initial state of the dialog box.

The Substitute dialog box has the same fields as the Search dialog box, plus a button for case-matching replacement and a text field for the replacement string.

/PATTERN
/NOPATTERN (D)
Enables or disables special interpretation of wildcard characters and a quoting character in the *search-string* parameter. You can use the SET SEARCH command to set the syntax for specifying a pattern to either VMS

style or ULTRIX style. For listing of VMS- and ULTRIX-style wildcards, see the /PATTERN qualifier on the SEARCH command.

When the /NOPATTERN qualifier is specified (or is the default), special interpretation of the asterisk, percent sign, and backslash characters is disabled.

/SINGLE
Specifies that only one occurrence of the search string is to be replaced with the replacement string. Specifying the /SINGLE qualifier causes LSE to perform a single substitution without prompting you for an action.

Parameters

search-string
Specifies the string for which to search.

replace-string
Specifies the string to substitute.

Description

The SUBSTITUTE command replaces one string of text with another. If the *search-string* and *replace-string* parameters appear on the command line, you should enclose each string in quotation marks. To obtain expected results, this is required if the search string contains (or you want the replacement string to contain) lowercase or nonalphanumeric characters.

If LSE prompts you for search and replace strings, you must omit any quotation marks that are not part of the text of the string.

LSE performs the search in the current direction. If you specify a null string for a search string, LSE uses the last search string specified in a SEARCH or SUBSTITUTE command. The SUBSTITUTE command differs from the SEARCH command in that, with the SUBSTITUTE command, LSE does not ignore an occurrence of the search string at the current cursor position.

When the substitution is complete, LSE leaves the cursor at the end of the last changed occurrence.

If you specify a repeat count, LSE ignores the count unless you specify the /SINGLE qualifier.

If the cursor is beyond the target of the search, LSE puts a message in the message buffer informing you that the target was not found.

DECwindows Interface Equivalent

SUBSTITUTE/DIALOG

Pull-down menu: $\left\{ \begin{array}{l} \text{Edit --> Replace . . .} \\ \text{Navigate --> Replace . . .} \end{array} \right\}$

Related Commands

SEARCH
SET SEARCH

Examples

1.
```
LSE>  SUBSTITUTE "man" "person"
```

Moves the cursor to the first occurrence of the word *man* in the current direction and invokes the confirmation prompt. A positive response replaces the word *man* with the word *person.*

2.
```
LSE>   SUBSTITUTE/CASE_MATCHING
_Search for:   str$append
_Replace with:   str$prefix
```

Moves the cursor to the first occurrence of the string *str$append* in the current direction. A positive response to the confirmation replaces *str$append* with *str$prefix.* If *str$append* occurs in uppercase (STR$APPEND), LSE puts the replacement string in uppercase (STR$PREFIX) even though you specified it in lowercase on the command line.

3.
```
LSE>  SUBSTITUTE/PATTERN "NAME_%_LENGTH" "NAME_B_LENGTH"
```

Moves the cursor to the next occurrence of a string consisting of *NAME_* and *_LENGTH* separated by any single character. A positive response to the confirmation prompt replaces that string with the string *NAME_B_LENGTH*.

TAB

Inserts indentation. If the cursor is at the beginning of the line, it moves to the current indentation level; otherwise, the cursor moves to the next tab stop.

Format

TAB

Description

The TAB command inserts blanks and tabs to move the cursor to the current indentation level (if at the beginning of the line) or to move the cursor to the next tab stop as set by the /TAB_INCREMENT qualifier on the DEFINE LANGUAGE command or by the SET TAB_INCREMENT command.

If the current indentation level is set to the beginning of the line and the cursor is at the beginning of the line, the TAB command inserts enough blank space to move the cursor to the first tab stop. In contrast, the ENTER TAB command has no effect when both the cursor and the current indentation level are at the beginning of the line.

Keypad Equivalent

Key	Keypad Mode
CTRL/I TAB	All

Related Commands

ENTER TAB
SET TAB_INCREMENT
UNTAB

TOGGLE SELECT_MARK

Sets and cancels the SELECT_MARK state.

Format

TOGGLE SELECT_MARK

Description

The TOGGLE SELECT_MARK command sets the select mark if it is not set, and cancels the select mark if it is set.

Keypad Equivalent

Key	Keypad Mode
E4 SELECT	EDT LK201, EVE LK201
KP7 SELECT	EVE VT100

Related Commands

CANCEL SELECT_MARK
SET SELECT_MARK

TWO WINDOWS

Splits the current window into two windows.

Format

TWO WINDOWS

Description

The TWO WINDOWS command splits the current window into two smaller windows. (This command is the same as the SPLIT WINDOW command, except it does not take a parameter.) You can view different buffers at the same time or different parts of the same buffer.

The cursor appears in the new lower window. Each window has its own status line and displays the buffer you are currently editing. To put a different buffer in the window, use one of the following commands:

GOTO BUFFER
GOTO FILE
NEXT BUFFER (if you have created more than one buffer)

To continue splitting windows, repeat the TWO WINDOWS command.

DECwindows Interface Equivalent

Pull-down menu: Display --> Split Window

Related Commands

CHANGE WINDOW_MODE
DELETE WINDOW
ENLARGE WINDOW
ONE WINDOW
OTHER WINDOW
PREVIOUS WINDOW
SET SCREEN
SHRINK WINDOW

Example

```
LSE> TWO WINDOWS
```

Splits the current window into two windows.

UNDO ENTER COMMENT

Reverses the effect of the last ENTER COMMENT command.

Format

UNDO ENTER COMMENT

Description

The UNDO ENTER COMMENT command deletes the comments created from pseudocode with the ENTER COMMENT command and restores the text to the pseudocode placeholders.

Related Commands

ENTER COMMENT

UNERASE

Restores the text deleted by the corresponding ERASE command that you most recently executed.

Format

UNERASE *[erase-option]*

Parameter

erase-option
The following are valid options with the UNERASE command:

CHARACTER
LINE
PLACEHOLDER
SELECTION
WORD

Description

The UNERASE command restores text erased by the previous ERASE CHARACTER, ERASE LINE, ERASE PLACEHOLDER, ERASE SELECTION, or ERASE WORD command. LSE inserts the restored text before the current cursor position, except for UNERASE PLACEHOLDER, which restores the text to its original position.

If you do not specify an erase option, LSE restores the text erased by the previous ERASE {CHARACTER, LINE, PLACEHOLDER, SELECTION, WORD} command, whichever one was the most recent.

The UNERASE PLACEHOLDER command also restores the placeholders created by the ENTER PSEUDOCODE command and erased by the ERASE PLACEHOLDER command.

Keypad Equivalent

UNERASE

Key	Keypad Mode
PF1-E2 INSERT HERE	EVE LK201

UNERASE CHARACTER

Key	Keypad Mode
PF1-keypad comma (,) UND C	EDT LK201, EDT VT100, EVE LK201
None	EVE VT100

UNERASE LINE

Key	Keypad Mode
PF1-PF4 UND L	EDT LK201, EDT VT100, EVE LK201
None	EVE VT100

UNERASE PLACEHOLDER

Key	Keypad Mode
PF1-CTRL/K	All

UNERASE SELECTION

Key	Keypad Mode
None	All

UNERASE WORD

Key	Keypad Mode
PF1-keypad minus (–) UND W	EDT LK201, EDT VT100, EVE LK201
None	EVE VT100

DECwindows Interface Equivalent

Pop-up menu: User Buffer --> Undo Erase xxx
Pull-down menu: Edit --> Undo Erase xxx

Note that *xxx* refers to whatever you last erased with the corresponding ERASE command.

Related Commands

ERASE CHARACTER
ERASE LINE
ERASE PLACEHOLDER
ERASE SELECTION
ERASE WORD

Example

```
LSE> UNERASE CHARACTER
```

Retrieves the contents of the deleted-character buffer.

UNEXPAND

Reverses the effect of the EXPAND command.

Format

UNEXPAND

Description

For LSE, the UNEXPAND command reverses the effect of the last EXPAND command. LSE deletes the range containing the text inserted as part of the last EXPAND command, and restores the token, placeholder, or alias that appeared at that position before the EXPAND command was issued.

Keypad Equivalent

Key	Keypad Mode
PF1-CTRL/E	EDT LK201, EDT VT100
PF1-CTRL//	EVE LK201, EVE VT100

Related Commands

EXPAND

UNRESERVE

Cancels the reservation of a CMS element with the same name and type as the input file for your current buffer.

Format

UNRESERVE

Description

The UNRESERVE command cancels the reservation in your current CMS library for an element with the same name and type as the input file for your current buffer. After successfully canceling a reservation, LSE deletes your current buffer and its corresponding file.

DECwindows Interface Equivalent

Pull-down menu: File --> Unreserve Element

Related Commands

REPLACE
RESERVE
SET CMS

UNTAB

Erases blanks and tabs to the left of the cursor, moving the cursor to the previous stop.

Format

UNTAB

Description

The UNTAB command removes blanks and tabs to the left of the cursor, moving the cursor to the previous tab stop set by the /TAB_INCREMENT qualifier on the DEFINE LANGUAGE command or by the SET TAB_INCREMENT command.

If no tabs or blanks immediately precede the cursor, this command has no effect. If nonblank or nontab characters are present in the column positions at or after the previous tab stop, LSE removes the blanks and tabs between those characters and the cursor, and then repositions the cursor after those characters, not at the tab stop.

Keypad Equivalent

Key	**Keypad Mode**
PF1-TAB	All

Related Commands

ENTER TAB
TAB

UPPERCASE WORD

Changes the current word to uppercase.

Format

UPPERCASE WORD

Description

The UPPERCASE WORD command puts the current word in uppercase letters. If the word is in both lowercase and uppercase letters, LSE changes all letters to uppercase.

If the cursor is between words, LSE puts the following word in uppercase letters. If a selected range is active, all the words within that range are changed to uppercase. Then, the cursor moves to the start of the next word.

DECwindows Interface Equivalent

Pull-down menu: Format --> Uppercase

Related Commands

CHANGE CASE
LOWERCASE WORD

VERIFY

Verifies that the specified SCA libraries are valid, and repairs any corrupted libraries.

Format

VERIFY *[library-spec[, . . .]]*

Qualifiers	**Defaults**
/[NO]LOG	/LOG
/[NO]RECOVER	/NORECOVER

Qualifiers

/LOG (D)
/NOLOG
Indicates whether SCA reports the library verification or repair operation.

/RECOVER
/NORECOVER (D)
Indicates whether SCA should repair a corrupted library.

If the interrupted command was a LOAD command, SCA deletes from the library any module that had begun to load but that had not completed loading. Also, SCA cannot recover modules that were waiting to be processed for loading when the interruption occurred. To load interrupted and waiting modules, issue a subsequent LOAD command and include those modules.

If the interrupted library operation was a DELETE MODULE command, the /RECOVER qualifier causes SCA to delete the incompletely deleted module. Any modules still waiting to be processed for deletion when the interruption occurred are excluded from the recovery operation; to delete them, you must respecify them in a subsequent DELETE MODULE command.

Parameter

library-spec[, . . .]
Specifies the SCA libraries to be verified. If you do not specify a library, SCA assumes you have specified the primary library.

Description

The VERIFY command performs the following operations to verify the validity of specific SCA libraries:

- Checks for corrupted libraries resulting from abnormal termination of a LOAD or DELETE MODULE command.
- Optionally, repairs corrupted libraries.

Example

```
SCA> VERIFY/RECOVER SCA$:[USER.SCA]
```

Determines if the library *SCA$:[USER.SCA]* has been corrupted and repairs any damage detected.

VIEW SOURCE

Displays an overview of the buffer.

Format

VIEW SOURCE

Qualifiers	**Defaults**
/DEBUG	
/DEPTH=*n*	/DEPTH=1

Qualifiers

/DEBUG
Provides a way to debug adjustment definitions by generating a copy of the source buffer indented as LSE views the indentation. LSE displays the result in a system buffer named $OVERVIEW, with all source lines visible. Numeric values for the indentation are also displayed, along with information about the adjustment applied to each line.

You cannot specify the /DEBUG qualifier with the /DEPTH qualifier.

/DEPTH=n
/DEPTH=1 (D)
Displays the top *n* levels of detail of the buffer. Lower levels are collapsed and represented by overview lines. If you specify /DEPTH=ALL, all the lines in the buffer are displayed; none of the lines are replaced by overview lines.

You cannot specify the /DEPTH qualifier with the /DEBUG qualifier.

Description

The VIEW SOURCE command displays the top *n* levels of detail of the whole buffer.

The editor determines the relative level of detail of a line by comparing the indentation of the line with the indentation of other lines. The editor's treatment of the indentation of a line is influenced by indentation adjustment definitions. For more information, see the DEFINE ADJUSTMENT command.

Keypad Equivalent

VIEW SOURCE/DEPTH=1

Key	Keypad Mode
PF1->	All

DECwindows Interface Equivalent

VIEW SOURCE/DEPTH=1
Pull-down menu: View --> Overview

VIEW SOURCE/DEPTH=ALL
Pull-down menu: View --> Source

Related Commands

COLLAPSE
DEFINE ADJUSTMENT
DEFINE LANGUAGE/OVERVIEW_OPTIONS
EXPAND
FOCUS

VIEW SOURCE

MODIFY LANGUAGE
SET NOOVERVIEW
SET OVERVIEW

WHAT LINE

Shows the current line number and total number of lines in the buffer. Also shows what percentage of the lines in the buffer are located above the current line.

Format

WHAT LINE

Description

The WHAT LINE command shows the current line number and total number of lines in the buffer. It also shows what percentage of the lines in the buffer are located above the current line.

This command is useful if you want to know whether to insert a page break or find out how many lines are in the buffer.

To move to a specific line by number, use the LINE command.

Related Commands

LINE

WRITE

Writes the contents of a buffer or the contents of the selected range to a file.

Format

WRITE *[file-spec]*

Qualifiers
/BUFFER=buffer-name
/DIALOG
/SELECT_RANGE
/VISIBLE

Qualifiers

/BUFFER=buffer-name
Indicates which buffer is to be written. The default is the current buffer.

/DIALOG
Instructs LSE to use a dialog box to prompt you for a parameter value. The command parameter is optional if you supply this qualifier. If you specify a command parameter with /DIALOG, LSE uses that parameter to set the initial state of the dialog box.

/SELECT_RANGE
Indicates that the selected range is to be written.

/VISIBLE
Indicates that the visible records in the buffer or selected range be written to a file. You must specify the *file-spec* parameter when you use this qualifier.

Parameter

file-spec
Specifies the file to which the buffer will be written. By default, LSE writes the data to the file associated with the buffer. This parameter is required if you specify the /SELECT_RANGE qualifier.

Description

The WRITE command places the contents of the indicated buffer in the file you specify. Your editing session continues until you issue an EXIT or QUIT command. If you are editing an existing file and do not supply a new file name, LSE creates a new version of that file when you issue the WRITE command.

When you issue a WRITE command without specifying a file name, LSE also displays an informational message and prompts you for confirmation before writing the buffer under either of the following conditions:

- If you have not modified the buffer (not made any changes during your editing session)
- If the buffer's status is read-only

If you issue the WRITE command and the current buffer is associated with a file of the same name, then LSE creates a new version of the file. If the buffer is unnamed, LSE prompts you for a name.

You may use the WRITE command and supply a file name at any time while you are in an editing session, thereby creating a new file that contains the output up to that point in your editing session. However, using the WRITE command to write the data to a different file does not change the file association of the buffer; that is, LSE still creates a new version of the file with the same name as that associated with the buffer when you exit from that editing session or subsequently use the WRITE command without specifying a file name. To change the file association, you use the SET OUTPUT_FILE command.

If you use the WRITE command to write to a directory that you have set read-only (using the SET DIRECTORY command), then LSE prompts you for confirmation before writing out the buffer.

DECwindows Interface Equivalent

WRITE
Pop-up menu: User buffer --> Save
Pull-down menu: File --> Save

WRITE/DIALOG
Pull-down menu: File --> Save as . . .

Related Commands

GOTO FILE
READ
SET OUTPUT_FILE

Example

```
LSE> WRITE/BUFFER=$SHOW SHOW.TXT
```

Causes LSE to write the current contents of the $SHOW buffer to a file called *SHOW.TXT*.

Appendix A

VAX Language-Sensitive Editor (LSE) Callable Interface

This appendix describes the LSE callable routines. It describes the purpose of the LSE callable routines, the parameters for a routine call, and the primary status returns. The parameter in the call syntax represents the object that you pass to an LSE routine. Each parameter description lists the data type and the passing mechanism for the object. The data types are standard VMS data types. The passing mechanism indicates how the parameter list is interpreted.

A.1 LSE Callable Routines

Callable LSE routines make LSE accessible from within other VAX languages and applications. You can call LSE from a program written in any VAX language that generates calls using the VAX Procedure Calling and Condition Handling Standard. You can also call LSE from VMS utilities, for example, MAIL. With callable LSE, you can perform text-processing functions within your program.

Callable LSE consists of a set of callable routines that reside in the LSE shareable image, LSESHR.EXE. You access callable LSE by linking to this shareable image, which includes the callable interface routine names and constants. As with the DCL-level LSE interface, you can use files for input to and output from callable LSE. You can also write your own routines for processing file input, output, and messages.

This chapter is written for system programmers familiar with the following:

- The VAX Procedure Calling and Condition Handling Standard
- The VMS Run-Time Library (RTL)

- The precise manner in which data types are represented on a VAX computer
- The method for calling routines written in a language other than the one you are using for the main program

The calling program must ensure that parameters passed to a called procedure, in this case LSE, are of the type and form that the LSE procedure accepts.

The LSE routines described in this chapter return condition values indicating the routine's completion status. When comparing a returned condition value with a test value, you should use the LIB$MATCH routine from the Run-Time Library. Do not test the condition value as if it were a simple integer.

A.1.1 Two Interfaces to Callable LSE

There are two interfaces that you can use to access callable LSE: the simplified callable interface and the full callable interface.

Simplified Callable Interface

The easiest way to use callable LSE is to use the simplified callable interface. LSE provides two alternative routines in its simplified callable interface. These routines in turn call additional routines that do the following:

- Initialize LSE
- Provide the editor with the parameters necessary for its operation
- Control the editing session
- Perform error handling

When using the simplified callable interface, you can use the LSE$LSE routine to specify a VMS command line for LSE, or you can call the LSE$EDIT routine to specify an input file and an output file. LSE$EDIT builds a command string that is then passed to the LSE$LSE routine. These two routines are described in detail in Section A.2.

If your application parses information that is not related to the operation of LSE, make sure the application obtains and uses all non-LSE parse information before the application calls the simplified callable interface. The reason is that the simplified callable interface destroys all parse information obtained and stored before the simplified callable interface was called.

If your application calls the DECwindows version of LSE, the application may call LSE$EDIT or LSE$LSE a single time only. Also, the application may not call XtInitialize before calling LSE. These restrictions will be lifted in a future version of LSE and DECwindows.

Full Callable Interface

The full callable interface consists of the main callable LSE routines and the LSE Utility routines.

To use the full callable interface, you have your program access the main callable LSE routines directly. These routines do the following:

- Initialize LSE (LSE$INTIALIZE)
- Execute LSE procedures (LSE$EXECUTE_INIFILE and LSE$EXECUTE_COMMAND)
- Give control to the editor (LSE$CONTROL)
- Terminate the editing session (LSE$CLEANUP)

When using the full callable interface, you must provide values for certain parameters. In some cases, the values you supply are actually addresses for additional routines. For example, when you call LSE$INITIALIZE, you must include the address of a routine that specifies initialization options. Depending on your particular application, you may also have to write additional routines. For example, you may need to write routines for performing file operations, handling errors, and otherwise controlling the editing session. Callable LSE provides utility routines that can perform some of these tasks for you. These utility routines do the following:

- Parse the VMS command line and build the item list used for initializing LSE
- Handle file operations
- Output error messages
- Handle conditions

If your application calls the DECwindows version of LSE, the application may call LSE$INITIALIZE a single time only. Also, the application may not call XtInitialize before calling LSE. These restrictions will be lifted in a future version of LSE and DECwindows.

Various topics relating to the full callable interface are discussed in the following sections:

- Section A.3 briefly describes the interface.
- Section A.3.1 describes the main callable LSE routines (LSE$INITIALIZE, LSE$EXECUTE_INIFILE, LSE$CONTROL, LSE$EXECUTE_COMMAND, and LSE$CLEANUP).
- Section A.3.2 discusses additional routines that LSE provides for use with the full callable interface.
- Section A.3.3 defines the requirements for routines that you can write for use with the full callable interface.

A.1.2 Shareable Image

Whether you use the simplified callable interface or the full callable interface, you access callable LSE by linking to the LSE shareable image, LSESHR.EXE. This image contains the routine names and constants available for use by an application. In addition, LSESHR.EXE provides the following symbols:

- TPU$GL_VERSION, the version of the shareable image
- TPU$GL_UPDATE, the update number of the shareable image
- TPU$_FACILITY, the VAXTPU facility code

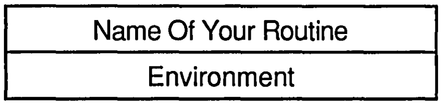
For more information about how to link to the shareable image LSESHR.EXE, refer to the *VMS System Services Reference Manual.*

A.1.3 Passing Parameters to Callable LSE Routines

Parameters are passed to callable LSE by reference or by descriptor. When the parameter is a routine, the parameter is passed by descriptor as a bound procedure value (BPV) data type.

A bound procedure value is a two-longword entity in which the first longword contains the address of a procedure entry mask, and the second longword is the environment value (see Figure A–1). The environment value is determined in a language-specific manner when the original bound procedure value is generated. When the bound procedure is called, the calling program loads the second longword into R1.

Figure A–1: Bound Procedure Value



ZK–4046–GE

A.1.4 Error Handling

When you use the simplified callable interface, LSE establishes its own condition handler, LSE$HANDLER, to handle all errors. When you use the full callable interface, there are two ways to handle errors:

- You can use LSE's default condition handler, LSE$HANDLER.
- You can write your own condition handler to process some of the errors, and you can call LSE$HANDLER to process the rest.

The default condition handler, LSE$HANDLER, is described in the routine description section of this chapter. For information about writing your own condition handler, see the *Introduction to VMS System Routines*.

A.1.5 Return Values

All LSE condition codes are declared as universal symbols. Therefore, you automatically have access to these symbols when you link your program to the shareable image. The condition code values are returned in R0.

Additional information about condition codes is provided in the descriptions of callable LSE routines found in subsequent sections. This information is provided under the section heading Condition Values Returned and indicates the values that are returned when the default condition handler is established.

A.2 Simplified Callable Interface

The LSE simplified callable interface consists of two routines: LSE$LSE and LSE$EDIT. These entry points to VAXTPU are useful for the following kinds of application:

- Those able to specify all the editing parameters on a single command line
- Those that need to specify only an input file and an output file

If your application parses information that is not related to the operation of LSE, make sure the application gets, and uses, all non-LSE parse information before the application calls the simplified callable interface. The simplified callable interface destroys all parse information obtained and stored before the simplified callable interface was called.

A.2.1 Example of the Simplified Interface

The following example calls LSE$EDIT to edit text in the file INFILE.DAT and writes the result to OUTFILE.DAT. Note that the parameters to LSE$EDIT must be passed by descriptor.

```
/*
  Sample C program that calls LSE.  This program uses LSE$EDIT to
  provide the names of the input and output files.
*/

#include descrip

int return_status;

static $DESCRIPTOR (input_file, "infile.dat");
static $DESCRIPTOR (output_file, "outfile.dat");

main (argc, argv)
    int argc;
    char *argv[];

    {
    /*
      Call LSE to edit text in "infile.dat" and write the result
      to "outfile.dat".  Return the condition code from LSE as the
      status of this program.
    */

    return_status = LSE$EDIT (&input_file, &output_file);
    exit (return_status);
    }
```

The next example performs the same task as the previous example. This time, the LSE$LSE entry point is used. LSE$LSE accepts a single argument, which is a command string starting with the verb LSEDIT. The command string can contain all of the qualifiers that are accepted by the LSEDIT command.

```
/*
  Sample C program that calls LSE.  This program uses LSE$LSE and
  specifies a command string
*/

#include descrip

int return_status;

static $DESCRIPTOR (command_prefix, "LSE/NOJOURNAL/NOCOMMAND/OUTPUT=");
static $DESCRIPTOR (input_file, "infile.dat");
static $DESCRIPTOR (output_file, "outfile.dat");
static $DESCRIPTOR (space_desc, " ");

char command_line [100];
static $DESCRIPTOR (command_desc, command_line);

main (argc, argv)
    int argc;
    char *argv[];

    {
    /*
      Build the command line for LSE.  Note that the command verb
      is LSEDIT.  The string we construct in the buffer command_line
      will be
        LSEDIT/NOJOURNAL/NOCOMMAND/OUTPUT=outfile.dat infile.dat
    */

    return_status = STR$CONCAT (&command_desc,
                                &command_prefix,
                                &output_file,
                                &space_desc,
                                &input_file);
    if (! return_status)
        exit (return_status);

    /*
      Now call LSE to edit the file
    */
    return_status = LSE$LSE (&command_desc);
    exit (return_status);
    }
```

A.3 Full Callable Interface

The LSE full callable interface consists of a set of routines that you can use to perform the following tasks:

- Specify initialization parameters
- Control file input/output
- Specify commands to be executed by LSE
- Control how conditions are handled

The individual LSE routines that perform these functions can be called from a user-written program.

This interface has two sets of routines: the main LSE callable routines and the LSE Utility routines. These LSE routines, and your own routines that pass parameters to the LSE routines, are the mechanism that your application uses to control LSE.

The following sections describe the main callable routines, how parameters are passed to these routines, the LSE Utility routines, and the requirements of user-written routines.

A.3.1 Main Callable LSE Utility Routines

This chapter describes the following callable LSE routines:

- LSE$INITIALIZE
- LSE$EXECUTE_INIFILE
- LSE$CONTROL
- LSE$EXECUTE_COMMAND
- LSE$CLEANUP

NOTE

Before calling any of these routines, you must establish LSE$HANDLER or provide your own condition handler. See the routine description of LSE$HANDLER at the end of this chapter and the section on the VAX Condition Handling Standard in the *Introduction to VMS System Routines* for information about establishing a condition handler.

A.3.2 Other LSE Utility Routines

The full callable interface includes several utility routines for which you can provide parameters. Depending on your application, you may be able to use these routines rather than write your own routines. These LSE Utility routines and their descriptions follow:

- LSE$CLIPARSE—Parses a command line and builds the item list for LSE$INITIALIZE.
- LSE$PARSEINFO—Parses a command and builds an item list for LSE$INITIALIZE.
- LSE$FILEIO—Is the default file I/O routine.
- LSE$MESSAGE—Writes error messages and strings by using the built-in procedure MESSAGE.
- LSE$HANDLER—Is the default condition handler.
- LSE$CLOSE_TERMINAL—Closes VAXTPU's channel to the terminal (and its associated mailbox) for the duration of a CALL_USER routine.

Note that LSE$CLIPARSE and LSE$PARSEINFO destroy the context maintained by the CLI$ routines for parsing commands.

A.3.3 User-Written Routines

This section defines the requirements for user-written routines. When these routines are passed to LSE, they must be passed as bound procedure values. (See Section A.1.3 for a description of bound procedure values.) Depending on your application, you may have to write one or all of the following routines:

- Routine for initialization callback

  This is a routine that LSE$INITIALIZE calls to obtain values for initialization parameters. The initialization parameters are returned as an item list.

- Routine for file I/O

  This is a routine that handles file operations. Instead of writing your own file I/O routine, you can use the LSE$FILEIO utility routine. LSE does not use this routine for journal file operations or for operations performed by the built-in procedure SAVE.

- Routine for condition handling

  This is a routine that handles error conditions. Instead of writing your own condition handler, you can use the default condition handler, LSE$HANDLER.

- Routine for the built-in procedure CALL_USER

  This is a routine that is called by the built-in procedure CALL_USER. You can use this mechanism to cause your program to get control during an editing session.

A.4 Examples of Using LSE Routines

Example A–1, Example A–2, Example A–3, and Example A–4 use callable LSE. The examples are included here for illustrative purposes only; Digital does not assume responsibility for supporting these examples.

Example A–1: Sample VAX BLISS Template for Callable VAXTPU

```
MODULE file_io_example (MAIN = top_level,
                        ADDRESSING_MODE (EXTERNAL = GENERAL)) =

BEGIN

FORWARD ROUTINE
    top_level,                  ! Main routine of this example
    lse_init,                   ! Initialize LSE
    lse_io;                     ! File I/O routine for LSE
!
! Declare the stream data structure passed to the file I/O routine
!
MACRO
    stream_file_id =  0, 0, 32, 0 % ,     ! File ID
    stream_rat =      6, 0,  8, 0 % ,     ! Record attributes
    stream_rfm =      7, 0,  8, 0 % ,     ! Record format
    stream_file_nm =  8, 0,  0, 0 % ;     ! File name descriptor

!
! Declare the routines that would actually do the I/O.  These must be supplied
! in another module
!
EXTERNAL ROUTINE
    my_io_open,                 ! Routine to open a file
    my_io_close,                ! Routine to close a file
    my_io_get_record,           ! Routine to read a record
    my_io_put_record;           ! Routine to write a record
```

(continued on next page)

Example A–1 (Cont.): Sample VAX BLISS Template for Callable VAXTPU

```
!
! Declare the LSE routines
!
EXTERNAL ROUTINE
    lse$fileio,                 ! LSE's internal file I/O routine
    lse$handler,                ! LSE's condition handler
    lse$initialize,             ! Initialize LSE
    lse$execute_inifile,        ! Execute the initial procedures
    lse$execute_command,        ! Execute an LSE statement
    lse$control,                ! Let user interact with LSE
    lse$cleanup;                ! Have LSE clean up after itself
!
! Declare the LSE literals
!
EXTERNAL LITERAL
    lse$k_close,                ! File I/O operation codes
    lse$k_close_delete,
    lse$k_open,
    lse$k_get,
    lse$k_put,

    lse$_access,                ! File access codes
    lse$k_io,
    lse$k_input,
    lse$k_output,

    lse$_calluser,             ! Item list entry codes
    lse$_fileio,
    lse$_outputfile,
    lse$_sectionfile,
    lse$_commandfile,
    lse$_filename,
    lse$_journalfile,
    lse$_options,

    lse$m_recover,              ! Mask for values in options bit vector
    lse$m_journal,
    lse$m_read,
    lse$m_command,
    lse$m_create,
    lse$m_section,
    lse$m_display,
    lse$m_output,

    lse$m_reset_terminal,       ! Masks for cleanup bit vector
    lse$m_kill_processes,
    lse$m_delete_exith,
    lse$m_last_time,
```

(continued on next page)

```
    tpu$_nofileaccess,          ! VAXTPU status codes
    tpu$_openin,
    tpu$_inviocode,
    tpu$_failure,
    tpu$_closein,
    tpu$_closeout,
    tpu$_readerr,
    tpu$_writeerr,
    tpu$_success;

ROUTINE top_level =

    BEGIN
!++
! Main entry point of your program
!--
! Your_initialization_routine must be declared as a BPV

    LOCAL
        initialize_bpv: VECTOR [2],
        status,
        cleanup_flags;
    !
    ! First establish the condition handler
    !
    ENABLE
        lse$handler ();
    !
    ! Initialize the editing session, passing LSE$INITIALIZE the address of
    ! the bound procedure value that defines the routine that LSE is
    ! to call to return the initization item list.
    !
    initialize_bpv [0] = lse_init;
    initialize_bpv [1] = 0;
    lse$initialize (initialize_bpv);
    !
    ! Call LSE to execute the contents of the command file, the debug file,
    ! or the LSE$INIT_PROCEDURE from the section file.
    !
    lse$execute_inifile();
    !
    ! Let LSE take over.
    !
    lse$control();
    !
    ! Have LSE cleanup after itself.
    !
    cleanup_flags = lse$m_reset_terminal OR     ! Reset the terminal
                    lse$m_kill_processes OR     ! Delete subprocesses
                    lse$m_delete_exith OR       ! Delete the exit handler
                    lse$m_last_time;            ! Last time calling the editor
```

(continued on next page)

```
    lse$cleanup (cleanup_flags);

    RETURN tpu$_success;

    END;

ROUTINE lse_init =

    BEGIN

    !
    ! Allocate the storage block needed to pass the file I/O routine as a
    ! bound procedure variable as well as the bit vector for the initialization
    ! options.
    !
    OWN
        file_io_bpv: VECTOR [2, LONG]
                     INITIAL (LSE_IO, 0),
        options;
    !
    ! These macros define the file names passed to LSE.
    !
    MACRO
        out_file = 'OUTPUT.TPU' % ,
        com_file = 'LSE$COMMAND' % ,
        sec_file = 'LSE$SECTION' % ,
        inp_file = 'FILE.TPU' % ;

    !
    ! Create the item list to pass to LSE.  Each item list entry consists of
    ! two words that specify the size of the item and its code, the address of
    ! the buffer containing the data, and a longword to receive a result (always
    ! zero, since LSE does not return any result values in the item list).
    !
    !               +---------------------------------+
    !               | Item Code        | Item Length  |
    !               +-----------------+---------------+
    !               |        Buffer Address           |
    !               +---------------------------------+
    !               |   Return Address (always 0)     |
    !               +---------------------------------+
    !
    ! Remember that the item list is always terminated with a longword containing
    ! a zero.
    !
    BIND
        item_list = UPLIT BYTE (
            WORD (4),                    ! Options bit vector
            WORD (lse$_options),
            LONG (options),
            LONG (0),
```

(continued on next page)

```
            WORD (4),                        ! File I/O routine
            WORD (lse$_fileio),
            LONG (file_io_bpv),
            LONG (0),

            WORD (%CHARCOUNT (out_file)),     ! Output file
            WORD (lse$_outputfile),
            LONG (UPLIT (%ASCII out_file)),
            LONG (0),

            WORD (%CHARCOUNT (com_file)),     ! Command file
            WORD (lse$_commandfile),
            LONG (UPLIT (%ASCII com_file)),
            LONG (0),

            WORD (%CHARCOUNT (sec_file)),    ! Section file
            WORD (lse$_sectionfile),
            LONG (UPLIT (%ASCII sec_file)),
            LONG (0),

            WORD (%CHARCOUNT (inp_file)),     ! Input file
            WORD (lse$_filename),
            LONG (UPLIT (%ASCII inp_file)),
            LONG (0),

            LONG (0));                       ! Terminating longword of 0
    !
    ! Initialize the options bitvector
    !
    options = lse$m_display OR               ! We have a display
              lse$m_section OR               ! We have a section file
              lse$m_create OR                ! Create a new file if one does not
                                             !   exist
              lse$m_command OR               ! We have a section file
              lse$m_output;                  ! We supplied an output file spec

    !
    ! Return the item list as the value of this routine for LSE to interpret.
    !
    RETURN item_list;

    END;                                     ! End of routine lse_init

ROUTINE lse_io (p_opcode, stream: REF BLOCK [ ,byte], data) =
!
! This routine determines how to process a TPU I/O request.
!
    BEGIN
```

(continued on next page)

Example A–1 (Cont.): Sample VAX BLISS Template for Callable VAXTPU

```
    LOCAL
        status;
!
! Is this one of ours, or do we pass it to LSE's file I/O routines?
!
    IF (..p_opcode NEQ lse$k_open) AND (.stream [stream_file_id] GTR 511)
    THEN
        RETURN lse$fileio (.p_opcode, .stream, .data);

!
! Either we're opening the file, or we know it's one of ours.
! Call the appropriate routine (not shown in this example).
!
    SELECTONE ..p_opcode OF
        SET

        [lse$k_open]:
            status = my_io_open (.stream, .data);

        [lse$k_close, lse$k_close_delete]:
            status = my_io_close (.stream, .data);

        [lse$k_get]:
            status = my_io_get_record (.stream, .data);

        [lse$k_put]:
            status = my_io_put_record (.stream, .data);

        [OTHERWISE]:
            status = tpu$_failure;

        TES;

    RETURN .status;

    END;                                        ! End of routine LSE_IO

END                                             ! End Module file_io_example
```

Example A–2: Normal LSE Setup in VAX FORTRAN

```
C       A sample FORTRAN program that calls LSE to act
C       normally, using the programmable interface.
C
C       IMPLICIT NONE

        INTEGER*4       CLEAN_OPT       !Options for clean up routine.
        INTEGER*4       STATUS          !Return status from LSE routines.
        INTEGER*4       BPV_PARSE(2)    !Set up a Bound Procedure Value.
        INTEGER*4       LOC_PARSE       !A local function call.
C
C       Declare the LSE functions.
C
        INTEGER*4       LSE$CONTROL
        INTEGER*4       LSE$CLEANUP
        INTEGER*4       LSE$EXECUTE_INIFILE
        INTEGER*4       LSE$INITIALIZE
        INTEGER*4       LSE$CLIPARSE
C
C       Declare a local copy to hold the values of LSE cleanup variables.
C
        INTEGER*4       RESET_TERMINAL
        INTEGER*4       DELETE_JOURNAL
        INTEGER*4       DELETE_BUFFERS,DELETE_WINDOWS
        INTEGER*4       DELETE_EXITH,EXECUTE_PROC
        INTEGER*4       PRUNE_CACHE,KILL_PROCESSES
        INTEGER*4       CLOSE_SECTION
C
C       Declare the LSE functions used as external.
C
        EXTERNAL        LSE$HANDLER
        EXTERNAL        LSE$CLIPARSE

        EXTERNAL        TPU$_SUCCESS    !External error message.

        EXTERNAL        LOC_PARSE       !User supplied routine to call LSE$CLIPARSE.
C
C       Declare the LSE cleanup variables as external.
C       These are the external literals that hold the
C       value of the options.
C
        EXTERNAL        LSE$M_RESET_TERMINAL
        EXTERNAL        LSE$M_DELETE_JOURNAL
        EXTERNAL        LSE$M_DELETE_BUFFERS,LSE$M_DELETE_WINDOWS
        EXTERNAL        LSE$M_DELETE_EXITH,LSE$M_EXECUTE_PROC
        EXTERNAL        LSE$M_PRUNE_CACHE,LSE$M_KILL_PROCESSES

100     CALL LIB$ESTABLISH ( LSE$HANDLER )       !Establish the condition handler.
C
C       Set up the Bound Procedure Value for the call to LSE$INITIALIZE.
C
        BPV_PARSE( 1 ) = %LOC( LOC_PARSE )
        BPV_PARSE( 2 ) = 0
```

(continued on next page)

```
C
C       Call the LSE initialization routine to do some setup work.
C
        STATUS = LSE$INITIALIZE ( BPV_PARSE )
C
C       Check the status.  If it is not a success, then signal the error.
C
        IF ( STATUS .NE. %LOC ( TPU$_SUCCESS ) ) THEN

                CALL LIB$SIGNAL( %VAL( STATUS ) )
                GOTO 9999

        ENDIF
C
C       Execute the LSE$_ init files and also a command file if it
C       was specified in the command line call to LSE.
C
        STATUS = LSE$EXECUTE_INIFILE ( )

        IF ( STATUS .NE. %LOC ( TPU$_SUCCESS ) ) THEN !Make sure everything is ok.

                CALL LIB$SIGNAL( %VAL( STATUS ) )
                GOTO 9999

        ENDIF
C
C       Invoke LSE as it normally would appear.
C
        STATUS = LSE$CONTROL ( )        !Call LSE.

        IF ( STATUS .NE. %LOC ( TPU$_SUCCESS ) ) THEN !Make sure everything is ok.

                CALL LIB$SIGNAL( %VAL( STATUS ) )
C               GOTO 9999
        ENDIF
C
C       Get the value of the option from the external literals.  In FORTRAN you
C       cannot use external literals directly so you must first get the value
C       of the literal from its external location.  Here we are getting the
C       values of the options that we want to use in the call to LSE$CLEANUP.
C
        DELETE_JOURNAL   = %LOC ( LSE$M_DELETE_JOURNAL )
        DELETE_EXITH     = %LOC ( LSE$M_DELETE_EXITH )
        DELETE_BUFFERS   = %LOC ( LSE$M_DELETE_BUFFERS )
        DELETE_WINDOWS   = %LOC ( LSE$M_DELETE_WINDOWS )
        EXECUTE_PROC     = %LOC ( LSE$M_EXECUTE_PROC )
        RESET_TERMINAL   = %LOC ( LSE$M_RESET_TERMINAL )
        KILL_PROCESSES   = %LOC ( LSE$M_KILL_PROCESSES )
        CLOSE_SECTION    = %LOC ( LSE$M_CLOSE_SECTION )
```

(continued on next page)

```
C
C       Now that we have the local copies of the variables we can do the
C       logical OR to set the multiple options that we need.
C
        CLEAN_OPT = DELETE_JOURNAL .OR. DELETE_EXITH .OR.
        1       DELETE_BUFFERS .OR. DELETE_WINDOWS .OR. EXECUTE_PROC
        1       .OR. RESET_TERMINAL .OR. KILL_PROCESSES .OR. CLOSE_SECTION
C
C       Do the necessary clean up.
C       LSE$CLEANUP wants the address of the flags as the parameter so
C       pass the %LOC of CLEAN_OPT, which is the address of the variable.

        STATUS = LSE$CLEANUP ( %LOC ( CLEAN_OPT ) )

        IF ( STATUS .NE. %LOC (TPU$_SUCCESS) ) THEN

                CALL LIB$SIGNAL( %VAL(STATUS) )

        ENDIF

9999    CALL LIB$REVERT         !Go back to normal processing -- handlers.

        STOP
        END
C
C
        INTEGER*4  FUNCTION LOC_PARSE

        INTEGER*4       BPV(2)          !A local bound procedure value

        CHARACTER*12    EDIT_COMM       !A command line to send to LSE$CLIPARSE
C
C       Declare the LSE functions used.
C
        INTEGER*4       LSE$FILEIO
        INTEGER*4       LSE$CLIPARSE
C
C       Declare this routine as external because it is never called directly and
C       we need to tell FORTRAN that it is a function and not a variable.
C
        EXTERNAL        LSE$FILEIO

        BPV(1) = %LOC(LSE$FILEIO)       !Set up the bound procedure value.
        BPV(2) = 0

        EDIT_COMM(1:12) = 'LSE TEST.TXT'
```

(continued on next page)

Example A–2 (Cont.): Normal LSE Setup in VAX FORTRAN

```
C
C       Parse the command line and build the item list for LSE$INITIALIZE.
C
9999     LOC_PARSE = LSE$CLIPARSE (EDIT_COMM, BPV , 0)

        RETURN
        END
```

Example A–3: Building a Callback Item List with VAX FORTRAN

```
       PROGRAM  TEST_LSE
C
       IMPLICIT NONE
C
C       Define the expected LSE return statuses.
C
       EXTERNAL        TPU$_SUCCESS
       EXTERNAL        TPU$_QUITTING
       EXTERNAL        TPU$_EXITING
C
C       Declare the LSE routines and symbols used.
C
       EXTERNAL        LSE$M_DELETE_CONTEXT
       EXTERNAL        LSE$HANDLER
       INTEGER*4       LSE$M_DELETE_CONTEXT
       INTEGER*4       LSE$INITIALIZE
       INTEGER*4       LSE$EXECUTE_INIFILE
       INTEGER*4       LSE$CONTROL
       INTEGER*4       LSE$CLEANUP
C
C      Use LIB$MATCH_COND to compare condition codes.
C
       INTEGER*4       LIB$MATCH_COND
C
C       Declare the external callback routine
C
       EXTERNAL        LSE_STARTUP       ! The LSE set-up function.
       INTEGER*4       LSE_STARTUP

       INTEGER*4       BPV(2)            ! Set up a bound procedure value.
```

(continued on next page)

```
C
C      Declare the functions used for working with the condition handler.
C
       INTEGER*4       LIB$ESTABLISH
       INTEGER*4       LIB$REVERT
C
C      Local flags and indices
C
       INTEGER*4       CLEANUP_FLAG       ! Flag(s) for LSE cleanup
       INTEGER*4       RET_STATUS
       INTEGER*4       MATCH_STATUS
C
C      Initializations
C
       RET_STATUS       = 0
       CLEANUP_FLAG     = %LOC(LSE$M_DELETE_CONTEXT)
C
C      Establish the default LSE condition handler.
C
       CALL LIB$ESTABLISH(%REF(LSE$HANDLER))
C
C      Set up the bound procedure value for the initialization callback.
C
       BPV(1)  =  %LOC (LSE_STARTUP)
       BPV(2)  =  0
C
C      Call the LSE procedure for initialization.
C
       RET_STATUS = LSE$INITIALIZE(BPV)

       IF (RET_STATUS .NE. %LOC(TPU$_SUCCESS)) THEN
       CALL LIB$SIGNAL (%VAL(RET_STATUS))
       ENDIF
C
C      Execute the LSE initialization file.
C
       RET_STATUS = LSE$EXECUTE_INIFILE()

       IF (RET_STATUS .NE. %LOC(TPU$_SUCCESS)) THEN
       CALL LIB$SIGNAL (%VAL(RET_STATUS))
       ENDIF
```

(continued on next page)

```
C
C      Pass control to LSE.
C
       RET_STATUS = LSE$CONTROL()
C
C      Test for valid exit condition codes.  You must use LIB$MATCH_COND
C      because the severity of TPU$_QUITTING can be set by the LSE
C      application.
C
       MATCH_STATUS = LIB$MATCH_COND (RET_STATUS, %LOC (TPU$_QUITTING),
       1                                          %LOC (TPU$_EXITING))
       IF (MATCH_STATUS .EQ. 0) THEN
       CALL LIB$SIGNAL (%VAL(RET_STATUS))
       ENDIF
C
C      Clean up after processing.
C
       RET_STATUS = LSE$CLEANUP(%REF(CLEANUP_FLAG))

       IF (RET_STATUS .NE. %LOC(TPU$_SUCCESS)) THEN
       CALL LIB$SIGNAL (%VAL(RET_STATUS))
       ENDIF
C
C      Set the condition handler back to the default.
C
       RET_STATUS = LIB$REVERT()

       END


       INTEGER*4 FUNCTION LSE_STARTUP

       IMPLICIT NONE

       INTEGER*4       OPTION_MASK        ! Temporary variable for LSE
       CHARACTER*44       SECTION_NAME    ! Temporary variable for LSE
C
C      External LSE routines and symbols.
C
       EXTERNAL        LSE$K_OPTIONS
       EXTERNAL        LSE$M_READ
       EXTERNAL        LSE$M_SECTION
       EXTERNAL        LSE$M_DISPLAY
       EXTERNAL        LSE$K_SECTIONFILE
       EXTERNAL        LSE$K_FILEIO
       EXTERNAL        LSE$FILEIO
       INTEGER*4       LSE$FILEIO
C
C      The bound procedure value used for setting up the file I/O routine.
C
       INTEGER*4       BPV(2)
```

(continued on next page)

```
C
C        Define the structure of the item list defined for the callback.
C
       STRUCTURE /CALLBACK/
       INTEGER*2       BUFFER_LENGTH
       INTEGER*2       ITEM_CODE
       INTEGER*4       BUFFER_ADDRESS
       INTEGER*4       RETURN_ADDRESS
       END STRUCTURE
C
C      There are a total of four items in the item list.
C
       RECORD /CALLBACK/ CALLBACK (4)
C
C      Make sure it is not optimized!
C
       VOLATILE /CALLBACK/
C
C       Define the options we want to use in the LSE session.
C
       OPTION_MASK = %LOC(LSE$M_SECTION) .OR. %LOC(LSE$M_READ)
       1       .OR. %LOC(LSE$M_DISPLAY)
C
C      Define the name of the initialization section file.
C
       SECTION_NAME = 'LSE$SECTION'
C
C      Set up the required I/O routine.  Use the LSE default.
C
       BPV(1) = %LOC(LSE$FILEIO)
       BPV(2) = 0
C
C      Build the callback item list.
C
C      Set up the edit session options.
C
       CALLBACK(1).ITEM_CODE = %LOC(LSE$K_OPTIONS)
       CALLBACK(1).BUFFER_ADDRESS = %LOC(OPTION_MASK)
       CALLBACK(1).BUFFER_LENGTH = 4
       CALLBACK(1).RETURN_ADDRESS = 0
C
C      Identify the section file to be used.
C
       CALLBACK(2).ITEM_CODE = %LOC(LSE$K_SECTIONFILE)
       CALLBACK(2).BUFFER_ADDRESS = %LOC(SECTION_NAME)
       CALLBACK(2).BUFFER_LENGTH = LEN(SECTION_NAME)
       CALLBACK(2).RETURN_ADDRESS = 0
```

(continued on next page)

Example A–3 (Cont.): Building a Callback Item List with VAX FORTRAN

```
C
C       Set up the I/O handler.
C
        CALLBACK(3).ITEM_CODE = %LOC(LSE$K_FILEIO)
        CALLBACK(3).BUFFER_ADDRESS = %LOC(BPV)
        CALLBACK(3).BUFFER_LENGTH = 4
        CALLBACK(3).RETURN_ADDRESS = 0
C
C       End the item list with zeros to indicate we are finished.
C
        CALLBACK(4).ITEM_CODE = 0
        CALLBACK(4).BUFFER_ADDRESS = 0
        CALLBACK(4).BUFFER_LENGTH = 0
        CALLBACK(4).RETURN_ADDRESS = 0
C
C       Return the address of the item list.
C
        LSE_STARTUP = %LOC(CALLBACK)

        RETURN
        END
```

Example A–4: Specifying a User-Written File I/O Routine in VAX C

```
/*
Simple example of a C program to invoke LSE.  This program provides its
own FILEIO routine instead of using the one provided by LSE.
*/
#include descrip
#include stdio

/* Data structures needed */

struct bpv_arg                      /* Bound procedure value */
    {
    int *routine_add ;              /* Pointer to routine */
    int env ;                       /* Environment pointer */
    } ;

struct item_list_entry              /* Item list data structure */
    {
    short int buffer_length;        /* Buffer length */
    short int item_code;            /* Item code */
    int *buffer_add;                /* Buffer address */
    int *return_len_add;            /* Return address */
    } ;
```

(continued on next page)

```
struct stream_type
    {
    int ident;                     /* Stream id */
    short int alloc;               /* File size */
    short int flags;               /* File record attributes/format */
    short int length;              /* Resultant file name length */
    short int stuff;               /* File name descriptor class & type */
    int nam_add;                   /* File name descriptor text pointer */
    } ;

globalvalue tpu$_success;          /* TPU Success code */
globalvalue tpu$_quitting;         /* Exit code defined by TPU */

globalvalue                        /* Cleanup codes defined by LSE */
    lse$m_delete_journal, lse$m_delete_exith,
    lse$m_delete_buffers, lse$m_delete_windows, lse$m_delete_cache,
    lse$m_prune_cache, lse$m_execute_file, lse$m_execute_proc,
    lse$m_delete_context, lse$m_reset_terminal, lse$m_kill_processes,
    lse$m_close_section, lse$m_delete_others, lse$m_last_time;
globalvalue                        /* Item codes for item list entries */
    lse$k_fileio, lse$k_options, lse$k_sectionfile,
    lse$k_commandfile ;
globalvalue                        /* Option codes for option item */
    lse$m_display, lse$m_section, lse$m_command, lse$m_create ;

globalvalue                        /* Possible item codes in item list */
    lse$_access, lse$_filename, lse$_defaultfile,
    lse$_relatedfile, lse$_record_attr, lse$_maximize_ver,
    lse$_flush, lse$_filesize;

globalvalue                        /* Possible access types for lse$_access */
    lse$k_io, lse$k_input, lse$k_output;

globalvalue                        /* RMS File Not Found message code */
    rms$_fnf;
globalvalue                        /* FILEIO routine functions */
    lse$k_open, lse$k_close, lse$k_close_delete,
    lse$k_get, lse$k_put;
int lib$establish ();              /* RTL routine to establish an event handler */
int lse$cleanup ();                /* LSE routine to free resources used */
int lse$control ();                /* LSE routine to invoke the editor */
int lse$execute_inifile ();        /* LSE routine to execute initialization code */
int lse$handler ();                /* LSE signal handling routine */
int lse$initialize ();             /* LSE routine to initialize the editor */
```

(continued on next page)

Example A–4 (Cont.): Specifying a User-Written File I/O Routine in VAX C

```
/*
   This function opens a file for either read or write access, based upon
   the item list passed as the data parameter.  Note that a full implementation
   of the file open routine would have to handle the default file, related
   file, record attribute, maximize version, flush and file size item code
   properly.
 */
open_file (data, stream)

int *data;
struct stream_type *stream;

{
    struct item_list_entry *item;
    char *access;                /* File access type */
    char filename[256];          /* Max file specification size */

    FILE *fopen();

    /* Process the item list */

    item = data;
    while (item->item_code != 0 && item->buffer_length != 0)
        {
        if (item->item_code == lse$_access)
            {
            if (item->buffer_add == lse$k_io) access = "r+";
            else if (item->buffer_add == lse$k_input) access = "r";
            else if (item->buffer_add == lse$k_output) access = "w";
            }
        else if (item->item_code == lse$_filename)
            {
            strncpy (filename, item->buffer_add, item->buffer_length);
            filename [item->buffer_length] = 0;
            lib$scopy_r_dx (&item->buffer_length, item->buffer_add,
                                                  &stream->length);
            }
        else if (item->item_code == lse$_defaultfile)
            {                           /* Add code to handle default file  */
            }                           /* spec here                        */
        else if (item->item_code == lse$_relatedfile)
            {                           /* Add code to handle related       */
            }                           /* file spec here                   */
        else if (item->item_code == lse$_record_attr)
            {                           /* Add code to handle record        */
            }                           /* attributes for creating files    */
        else if (item->item_code == lse$_maximize_ver)
            {                           /* Add code to maximize version     */
            }                           /* number with existing file here   */
        else if (item->item_code == lse$_flush)
            {                           /* Add code to cause each record    */
            }                           /* to be flushed to disk as written */
```

(continued on next page)

```
        else if (item->item_code == lse$_filesize)
            {                          /* Add code to handle specification */
            }                          /* of initial file allocation here  */
        ++item;         /* get next item */
        }
    stream->ident = fopen(filename,access);
    if (stream->ident != 0)
        return tpu$_success;
    else
        return rms$_fnf;
}
/*
  This procedure closes a file.
 */
close_file (data,stream)
struct stream_type *stream;

{
    close(stream->ident);
    return tpu$_success;
}
/*
  This procedure reads a line from a file.
 */
read_line(data,stream)
struct dsc$descriptor *data;
struct stream_type *stream;

{
    char textline[984];                /* Max line size for TPU records */
    int len;

    globalvalue rms$_eof;              /* RMS End-Of-File code */

    if (fgets(textline,984,stream->ident) == NULL)
        return rms$_eof;
    else
        {
        len = strlen(textline);
        if (len > 0)
            len = len - 1;
        return lib$scopy_r_dx (&len, textline, data);
        }
}
```

(continued on next page)

Example A–4 (Cont.): Specifying a User-Written File I/O Routine in VAX C

```
/*
  This procedure writes a line to a file.
 */
write_line(data,stream)
struct dsc$descriptor *data;
struct stream_type *stream;

{
    char textline[984];                  /* Max line size for TPU records */

    strncpy (textline, data->dsc$a_pointer, data->dsc$w_length);
    textline [data->dsc$w_length] = 0;
    fputs(textline,stream->ident);
    fputs("\n",stream->ident);
    return tpu$_success;
}
/*
   This procedure will handle I/O for LSE.
 */
fileio(code,stream,data)
int *code;
int *stream;
int *data;

{
    int status;

/* Dispatch based on code type.  Note that a full implementation of the      */
/* file I/O routines would have to handle the close and delete code properly */
/* instead of simply closing the file.                                        */

    if (*code == lse$k_open)                    /* Initial access to file */
        status = open_file (data,stream);
    else if (*code == lse$k_close)              /* End access to file */
        status = close_file (data,stream);
    else if (*code == lse$k_close_delete)       /* Treat same as close */
        status = close_file (data,stream);
    else if (*code == lse$k_get)                /* Read a record from a file */
        status = read_line (data,stream);
    else if (*code == lse$k_put)                /* Write a record to a file */
        status = write_line (data,stream);
    else
        {                                       /* Who knows what we got? */
  status = tpu$_success;
        printf ("Bad FILEIO I/O function requested");
        }
    return status;
}
```

(continued on next page)

```
/*
   This procedure formats the initialization item list and returns it as
   a return value.
 */
callrout()
{
    static struct bpv_arg add_block =
        { fileio, 0 } ;         /* BPV for fileio routine */
    int options ;
    char *section_name = "LSE$SECTION";
    static struct item_list_entry arg[] =
        {/* length code              buffer add return add */
               { 4,lse$k_fileio,     0,         0 },
               { 4,lse$k_options,    0,         0 },
               { 0,lse$k_sectionfile,0,         0 },
               { 0,0,                0,         0 }
        };

    /* Setup file I/O routine item entry */
    arg[0].buffer_add = &add_block;

    /* Setup options item entry.  Leave journaling off. */
    options = lse$m_display | lse$m_section;
    arg[1].buffer_add = &options;

    /* Setup section file name */
    arg[2].buffer_length = strlen(section_name);
    arg[2].buffer_add = section_name;

    return arg;
}

/*
   Main program.  Initializes LSE, then passes control to it.
 */
main()
{
    int return_status ;
    int cleanup_options;
    struct bpv_arg add_block;

/* Establish as condition handler the normal LSE handler */

    lib$establish(lse$handler);

/* Setup a BPV to point to the callback routine */

    add_block.routine_add = callrout ;
    add_block.env = 0;

/* Do the initialize of LSE */

    return_status = lse$initialize(&add_block);
    if (!return_status)
        exit(return_status);
```

(continued on next page)

Example A–4 (Cont.): Specifying a User-Written File I/O Routine in VAX C

```
/* Have LSE execute the procedure LSE$INIT_PROCEDURE from the section file */
/* and then compile and execute the code from the command file */

    return_status = lse$execute_inifile();
    if (!return_status)
        exit (return_status);

/* Turn control over to LSE */

    return_status = lse$control ();
    if (!return_status)
        exit(return_status);

/* Now clean up. */

    cleanup_options = lse$m_last_time | lse$m_delete_context;
    return_status = lse$cleanup (&cleanup_options);
    exit (return_status);

    printf("Experiment complete");
}
```

A.5 LSE Routines

The following pages describe the individual LSE routines.

LSE$CLEANUP—Free System Resources Used During LSE Session

Cleans up internal data structures, frees memory, and restores terminals to their initial state.

This is the final routine called in each interaction with LSE.

Format

LSE$CLEANUP *flags*

Returns

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

Longword condition value. Most utility routines return a condition value in R0. The condition value that this routine can return is listed in the Condition Value Returned section.

Argument

flags

VMS Usage:	**mask_longword**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Flags (or mask) defining the cleanup options. The *flags* argument is the address of a longword bit mask defining the cleanup options or the address of a 32-bit mask defining the cleanup options. This mask is the logical OR of the flag bits you want to set. LSE$V . . . indicates a bit item and

LSE$M . . . indicates a mask. Table A–1 describes the various cleanup options.

Table A–1: LSE$CLEANUP Options

Symbol[1]	Function
LSE$M_DELETE_JOURNAL	Closes and deletes the journal file if it is open.
LSE$M_DELETE_EXITH	Deletes LSE's exit handler.
LSE$M_DELETE_BUFFERS	Deletes all text buffers. If this is not the last time you are calling LSE, then all variables referring to these data structures are reset as if by the built-in procedure DELETE. If a buffer is deleted, then all ranges and markers within that buffer, and any subprocesses using that buffer, are also deleted.
LSE$M_DELETE_WINDOWS	Deletes all windows. If this is not the last time you are calling LSE, then all variables referring to these data structures are reset as if by the built-in procedure DELETE.
LSE$M_DELETE_CACHE	Deletes the virtual file manager's data structures and caches. If this deletion is requested, then all buffers are also deleted. If the cache is deleted, the initialization routine has to reinitialize the virtual file manager the next time it is called.
LSE$M_PRUNE_CACHE	Frees up any virtual file manager caches that have no pages allocated to buffers. This frees up any caches that may have been created during the session but that are no longer needed.
LSE$M_EXECUTE_FILE	Reexecutes the command file if LSE$EXECUTE_INIFILE is called again. You must set this bit if you plan to specify a new file name for the command file. This option is used in conjunction with the option bit passed to LSE$INITIALIZE indicating the presence of the /COMMAND qualifier.

[1]The prefix can be LSE$M_ or LSE$V_. LSE$M_ denotes a mask corresponding to the specific field in which the bit is set. LSE$V_ is a bit number.

(continued on next page)

Table A–1 (Cont.): LSE$CLEANUP Options

Symbol[1]	Function
LSE$M_EXECUTE_PROC	Looks up LSE$INIT_PROCEDURE and executes it the next time LSE$EXECUTE_INIFILE is called.
LSE$M_DELETE_CONTEXT	Deletes the entire context of LSE. If this option is specified, then all other options are implied, except for executing the initialization file and initialization procedure.
LSE$M_RESET_TERMINAL	Resets the terminal to the state it was in upon entry to LSE. The terminal mailbox and all windows are deleted. If the terminal is reset, then it is reinitialized the next time LSE$INITIALIZE is called.
LSE$M_KILL_PROCESSES	Deletes all subprocesses created during the session.
LSE$M_CLOSE_SECTION[2]	Closes the section file and releases the associated memory. All buffers, windows, and processes are deleted. The cache is purged and the flags are set for reexecution of the initialization file and initialization procedure. If the section is closed and if the option bit indicates the presence of the /SECTION qualifier, then the next call to LSE$INITIALIZE attempts a new restore operation.
LSE$M_DELETE_OTHERS	Deletes all miscellaneous preallocated data structures. Memory for these data structures is reallocated the next time LSE$INITIALIZE is called.
LSE$M_LAST_TIME	This bit should be set only when you are calling LSE for the last time. Note that if you set this bit and then recall LSE, the results are unpredictable.

[1]The prefix can be LSE$M_ or LSE$V_. LSE$M_ denotes a mask corresponding to the specific field in which the bit is set. LSE$V_ is a bit number.
[2]Using the simplified callable interface does not set LSE$_CLOSE_SECTION. This feature allows you to make multiple calls to LSE$LSE without requiring you to open and close the section file on each call.

Condition Value Returned

TPU$_SUCCESS	Normal successful completion.

Description

The LSE$CLEANUP routine is the final routine called in each interaction with LSE. It tells LSE to clean up its internal data structures and to prepare for additional invocations. You can control what this routine resets by setting or clearing the flags described previously.

When you finish with LSE, call this routine to free the memory and restore the characteristics of the terminal to their original settings.

If you intend to exit after calling LSE$CLEANUP, do not delete the data structures; VMS does this automatically. Allowing VMS to delete the structures improves the performance of your program.

Notes

1. When you use the simplified interface, LSE automatically sets the following flags:
    - LSE$V_RESET_TERMINAL
    - LSE$V_DELETE_BUFFERS
    - LSE$V_DELETE_JOURNAL
    - LSE$V_DELETE_WINDOWS
    - LSE$V_DELETE_EXITH
    - LSE$V_EXECUTE_PROC
    - LSE$V_EXECUTE_FILE
    - LSE$V_PRUNE_CACHE
    - LSE$V_KILL_PROCESSES
2. If this routine does not return a success status, no other calls to the editor should be made.

LSE$CLIPARSE—Parse a Command Line

Parses a command line and builds the item list for LSE$INITIALIZE.

It calls CLI$DCL_PARSE to establish a command table and a command to parse. It then calls LSE$PARSEINFO to build an item list for LSE$INITIALIZE.

If your application parses information that is not related to the operation of LSE, make sure the application gets, and uses, all non-LSE parse information before the application calls LSE$CLIPARSE. LSE$CLIPARSE destroys all parse information obtained and stored before LSE$CLIPARSE was called.

Format

LSE$CLIPARSE *string, fileio, call_user*

Returns

VMS Usage:	**item_list**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

This routine returns the address of an item list.

Arguments

string

VMS Usage:	**char_string**
type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

Command line. The *string* argument is the address of a descriptor of an LSE command.

fileio

VMS Usage:	**vector_longword_unsigned**
type:	**bound procedure value**
access:	**read only**
mechanism:	**by descriptor**

File I/O routine. The *fileio* argument is the address of a descriptor of a file I/O routine.

call_user

VMS Usage:	**vector_longword_unsigned**
type:	**bound procedure value**
access:	**read only**
mechanism:	**by descriptor**

Call-user routine. The *call_user* argument is the address of a descriptor of a call-user routine.

LSE$CLOSE_TERMINAL—Close Channel to Terminal

Closes LSE's channel to the terminal.

Format

LSE$CLOSE_TERMINAL

Returns

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

Longword condition value. Most utility routines return a condition value in R0. The condition value that this routine can return is listed in the Condition Value Returned section.

Condition Value Returned

TPU$_SUCCESS	Normal successful completion.

Description

The LSE$CLOSE routine is used with the built-in procedure CALL_USER and its associated call-user routine to control LSE's access to the terminal. When a call-user routine invokes LSE$CLOSE_TERMINAL, LSE closes its channel to the terminal and the channel of LSE's associated mailbox.

When the call-user routine returns control to it, LSE automatically reopens a channel to the terminal and redisplays the visible windows.

A call-user routine can use LSE$CLOSE_TERMINAL at any point in the program and as many times as necessary. If the terminal is already closed to LSE when LSE$CLOSE_TERMINAL is used, the call is ignored.

LSE$CONTROL—Pass Control to LSE

Is the main processing routine of LSE. It is responsible for reading the text and commands and executing them. When you call this routine (after calling LSE$INITIALIZE), control is turned over to LSE.

Format

LSE$CONTROL *(last-line, last-char, out-file)*

Returns

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

Longword condition value. Most utility routines return a condition value in R0. Condition values that this routine can return are listed in the Condition Values Returned section.

Arguments

last-line

VMS Usage:	**integer**
type:	**longword (signed)**
access:	**write only**
mechanism:	**by reference**

A signed longword to receive the final position in buffer LSE$MAIN_BUFFER. The first line in the file is line 1.

Arguments

integer

VMS Usage:	**integer**
type:	**longword (signed)**
access:	**write only**
mechanism:	**by reference**

A signed longword to receive the final column position in buffer LSE$MAIN_BUFFER. The first column on a line is column 1.

Arguments

char-string

VMS Usage:	**char-string**
type:	**character string**
access:	**write only**
mechanism:	**by descriptor**

A character string that receives the file specification of the file to which the buffer, pointed to by the VAXTPU variable LSE$MAIN_BUFFER, was written upon exit. If LSE$MAIN_BUFFER was not written to its designated output file, the *out-file* is the file specification of the file read into LSE$MAIN_BUFFER. If you issue the QUIT command, this specification is the null string. You can use this information to return to this file during a subsequent edit. LSE uses STR$COPY to fill in this string.

The *last-line* and *last-char* arguments together describe the last current position in the buffer pointed to by the VAXTPU variable LSE$MAIN_BUFFER. You can use this information to return to this file position in a subsequent edit.

Condition Values Returned

TPU$_EXITING	A result of EXIT (when the default condition handler is enabled).
TPU$_QUITTING	A result of QUIT (when the default condition handler is enabled).
TPU$_RECOVERFAIL	A recovery operation was terminated abnormally.

Description

The LSE$CONTROL routine controls the edit session. It is responsible for reading the text and commands and executing them. Windows on the screen are updated to reflect the edits that are performed.

LSE$EDIT—Edit a File

Builds a command string from its parameters and passes it to the LSE$LSE routine.

LSE$EDIT is another entry point to LSE's simplified callable interface.

Format

LSE$EDIT *input, output*

Returns

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

Longword condition value. Most utility routines return a condition value in R0. Condition values that this routine can return are listed in the Condition Values Returned section.

Arguments

input

VMS Usage:	**char_string**
type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

Input file name. The *input* argument is the address of a descriptor of a file specification.

output
VMS Usage: **char_string**
type: **character string**
access: **read only**
mechanism: **by descriptor**

Output file name. The *output* argument is the address of a descriptor of an output file specification. It is used with the /OUTPUT command qualifier.

Condition Values Returned

The LSE$EDIT routine returns any value returned by LSE$LSE.

Description

The LSE$EDIT routine builds a command string and passes it to LSE$LSE. If the length of the output string is greater than 0, you can include it in the command line by using the /OUTPUT qualifier, as follows:

```
LSEDIT [/OUTPUT= output] input
```

If your application parses information that is not related to the operation of LSE, make sure the application gets, and uses, all non-LSE parse information before the application calls LSE$EDIT. LSE$EDIT destroys all parse information obtained and stored before LSE$EDIT is called.

LSE$EXECUTE_COMMAND—Execute One or More VAXTPU Statements

Allows your program to execute VAXTPU statements.

Format

LSE$EXECUTE_COMMAND *string*

Returns

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

Longword condition value. Most utility routines return a condition value in R0. Condition values that this routine can return are listed in the Condition Values Returned section.

Argument

string

VMS Usage:	**char_string**
type:	**character string**
access:	**read only**
mechanism:	**by value**

VAXTPU statement. The *string* argument is the address of a descriptor of a character string denoting one or more VAXTPU statements.

Condition Values Returned

TPU$_SUCCESS	Normal successful completion.
TPU$_EXITING	EXIT built-in procedure was invoked.
TPU$_QUITTING	QUIT built-in procedure was invoked.
TPU$_EXECUTEFAIL	Execution aborted. This could be because of execution errors or compilation errors.

Description

The LSE$EXECUTE_COMMAND routine performs the same function as the built-in procedure EXECUTE described in the *VAX Text Processing Utility Manual.*

LSE$EXECUTE_INIFILE—Execute Initialization Files

Allows you to execute a user-written initialization file.

This routine must be executed after the editor is initialized, but before any other commands are processed.

Format

LSE$EXECUTE_INIFILE

Returns

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

Longword condition value. Most utility routines return a condition value in R0. Condition values that this routine can return are listed in the Condition Values Returned section.

Condition Values Returned

TPU$_SUCCESS	Normal successful completion.
TPU$_EXITING	A result of EXIT. If the default condition handler is being used, the session is terminated.
TPU$_QUITTING	A result of QUIT. If the default condition handler is being used, the session is terminated.
TPU$_COMPILEFAIL	The compilation of the initialization file was unsuccessful.

TPU$_EXECUTEFAIL	The execution of the statements in the initialization file was unsuccessful.
TPU$_FAILURE	General code for all other errors.

Description

The LSE$EXECUTE_INIFILE routine causes VAXTPU to perform the following steps:

1. The command file is read into a buffer. If you specify a file on the command line that cannot be found, an error message is displayed and the routine is aborted. The default is LSE$COMMAND.TPU.
2. If you specify the /DEBUG qualifier on the command line, the DEBUG file is read into a buffer. The default is SYS$SHARE:LSE$DEBUG.TPU.
3. The DEBUG file is compiled and executed (if available).
4. TPU$INIT_PROCEDURE is executed (if available).
5. The command buffer is compiled and executed (if available).
6. TPU$INIT_POSTPROCEDURE is executed (if available).

NOTE

If you call this routine after calling LSE$CLEANUP, you must set the flags LSE$V_EXECUTE_PROC and LSE$V_EXECUTE_FILE. Otherwise, the initialization file does not execute.

LSE$FILEIO—Perform File Operations

Handles all LSE file operations. Your own file I/O routine can call this routine to perform some operations for it. However, the routine that opens the file must perform *all* operations for that file. For example, if LSE$FILEIO opens the file, it must also close it.

Format

LSE$FILEIO *code, stream, data*

Returns

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

Longword condition value. Most utility routines return a condition value in R0. Condition values that this routine can return are listed in the Condition Values Returned section.

Arguments

code

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Item code specifying an LSE function. The *code* argument is the address of a longword containing an item code from LSE specifying a function to perform. Following are the item codes that you can specify in the file I/O routine:

- LSE$K_OPEN

  This item code specifies that the data parameter is the address of an item list. This item list contains the information necessary to open the file. The stream parameter should be filled in with a unique identifying value to be used for all future references to this file. The resultant file name should also be copied with a dynamic string descriptor.

- LSE$K_CLOSE

  The file specified by the *stream* argument is to be closed. All memory being used by its structures can be released.

- LSE$K_CLOSE_DELETE

  The file specified by the *stream* argument is to be closed and deleted. All memory being used by its structures can be released.

- LSE$K_GET

  The data parameter is the address of a dynamic string descriptor to be filled with the next record from the file specified by the *stream* argument. The routine should use the routines provided by the VMS Run-Time Library to copy text into this descriptor. LSE frees the memory allocated for the data read when the file I/O routine indicates that the end of the file has been reached.

- LSE$K_PUT

  The *data* parameter is the address of a descriptor for the data to be written to the file specified by the *stream* argument.

stream

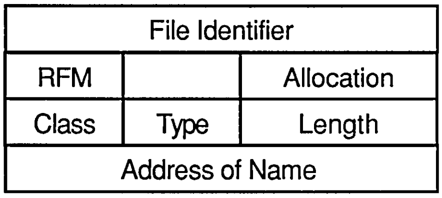
VMS Usage:	**unspecified**
type:	**longword (unsigned)**
access:	**modify**
mechanism:	**by reference**

File description. The *stream* argument is the address of a data structure consisting of four longwords. This data structure is used to describe the file to be manipulated.

This data structure is used to refer to all files. It is written to when an open file request is made. All other requests use information in this structure to determine which file is being referenced.

Figure A–2 shows the stream data structure.

Figure A–2: Stream Data Structure



The first longword is used to hold a unique identifier for each file. The user-written file I/O routine is restricted to values between 0 and 511. Thus, you can have up to 512 files open simultaneously.

The second longword is divided into three fields. The low word is used to store the allocation quantity, that is, the number of blocks allocated to this file from the FAB (FAB$L_ALQ). This value is used later to calculate the output file size for preallocation of disk space. The low-order byte of the second word is used to store the record attribute byte (FAB$B_RAT) when an existing file is opened. The high-order byte is used to store the record format byte (FAB$B_RFM) when an existing file is opened. The values in the low word and the low-order and high-order bytes of the second word are used for creating the output file in the same format as the input file. These three fields are to be filled in by the routine opening the file.

The last two longwords are used as a descriptor for the resultant or the expanded file name. This name is used later when LSE processes EXIT commands. This descriptor is to be filled in with the file name after an open operation. It should be allocated with either the routine LIB$SCOPY_R_DX or the routine LIB$SCOPY_DX from the Run-Time Library. This space is freed by LSE when it is no longer needed.

data

VMS Usage:	**item_list_3**
type:	**longword (unsigned)**
access:	**modify**
mechanism:	**by reference**

Stream data. The *data* argument is either the address of an item list or the address of a descriptor.

NOTE

The meaning of this parameter depends on the item code specified in the code field.

When the LSE$K_OPEN item code is issued, the data parameter is the address of an item list containing information about the open request. The following LSE item codes are available for specifying information about the open request:

- LSE$_ACCESS allows you to specify one of three item codes in the buffer address field, as follows:
  - LSE$K_IO
  - LSE$K_INPUT
  - LSE$K_OUTPUT
- LSE$_FILENAME is used for specifying the address of a string to use as the name of the file you are opening. The length field contains the length of this string, and the address field contains the address.
- LSE$_DEFAULTFILE is used for assigning a default file name to the file being opened. The buffer-length field contains the length, and the buffer-address field contains the address of the default file name.
- LSE$_RELATEDFILE is used for specifying a related file name for the file being opened. The buffer-length field contains the length, and the buffer-address field contains the address of a string to use as the related file name.
- LSE$_RECORD_ATTR specifies that the buffer-address field contains the value for the record attribute byte in the FAB (FAB$B_RAT) used for file creation.
- LSE$_RECORD_FORM specifies that the buffer-address field contains the value for the record format byte in the FAB (FAB$B_RFM) used for file creation.
- LSE$_MAXIMIZE_VER specifies that the version number of the output file should be one higher than the highest existing version number.
- LSE$_FLUSH specifies that the file should have every record flushed after it is written.

- LSE$_FILESIZE is used for specifying a value to be used as the allocation quantity when creating the file. The value is specified in the buffer-address field.
- LSE$_EOF_BLOCK specifies the end-of-file block number of the file. The BUFADR field is the address of a longword into which the file I/O routine must write the file's end-of-file block number (from XAB$L_EBK in $XABFHC).
- LSE$_EOF_FFB specifies the file's first free byte offset into the end-of-file block. The BUFADR field is the address of a word into which the file I/O routine must write the file's first free byte offset into the end-of-file block (XAB$W_FFB in $XABFHC).

Condition Values Returned

The LSE$FILEIO routine returns an RMS status code to LSE. The file I/O routine is responsible for signaling all errors if you want any messages displayed.

Description

By default, LSE$FILEIO creates variable-length files with carriage-return record attributes (FAB$B_RFM = VAR, FAB$B_RAT = CR). If you pass to it the LSE$_RECORD_ATTR or LSE$_RECORD_FORM item, that item is used instead. The following combinations of formats and attributes are acceptable:

Format	Attributes
STM,STMLF,STMCR	0,BLK,CR,BLK+CR
VAR	0,BLK,FTN,CR,BLK+FTN,BLK+CR

All other combinations are converted to VAR format with CR attributes.

This routine always puts values greater than *511* in the first longword of the stream data structure. Because a user-written file I/O routine is restricted to the values *0* through *511*, you can easily distinguish the file-control blocks (FCB) this routine fills in from the ones you created.

NOTE

LSE uses LSE$FILEIO by default when you use the simplified callable interface. When you use the full callable interface, you must explicitly invoke LSE$FILEIO or provide your own file I/O routine.

LSE$HANDLER—LSE Condition Handler

Is LSE's condition handler.

The LSE condition handler invokes the Put Message (SYS$PUTMSG) system service, passing it the address of LSE$MESSAGE.

Format

LSE$HANDLER *signal_vector, mechanism_vector*

Returns

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

Longword condition value.

Arguments

signal_vector

VMS Usage:	**arg_list**
type:	**longword (unsigned)**
access:	**modify**
mechanism:	**by reference**

Signal vector. See the *VMS System Services Reference Manual* for information about the signal vector passed to a condition handler.

mechanism_vector

VMS Usage:	**arg_list**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Mechanism vector. See the *VMS System Services Reference Manual* for information about the mechanism vector passed to a condition handler.

Description

The LSE$MESSAGE routine performs the actual output of the message. The Put Message (SYS$PUTMSG) system service formats only the message. It gets the settings for the message flags and facility name from the variables described in Section A.1.2. You must use the VAXTPU built-in procedure SET to modify those values.

If the condition value received by the handler has a fatal status or does not have an LSE, VAXTPU, CLI$, or SCA facility code, the condition is resignaled.

If the condition is TPU$_QUITTING, TPU$_EXITING, or TPU$_RECOVERFAIL, a request to unwind is made to the establisher of the condition handler.

After handling the message, the condition handler returns with a continue status. VAXTPU error message requests are made by signaling a condition to indicate which message should be written out. The arguments in the signal array are a correctly formatted message argument vector. This vector sometimes contains multiple conditions and formatted ASCII output (FAO) arguments for the associated messages. For example, if the editor attempts to open a file that does not exist, the VAXTPU message TPU$_NOFILEACCESS is signaled. The FAO argument to this message is a string for the name of the file. This condition has an error status, followed by the VMS RMS status field (STS) and status-value field (STV). Because this condition does not have a fatal severity, LSE continues after handling the error.

The editor does not automatically return from LSE$CONTROL. If you call the LSE$CONTROL routine, you must explicitly establish a way to regain control (for example, using the built-in procedure CALL_USER). Also, if you establish your own condition handler but call the LSE handler for certain conditions, the default condition handler *must* be established at the point in your program where you want to return control.

See the *Introduction to VMS System Routines* for information about the VAX Condition Handling Standard.

LSE$INITIALIZE—Initialize VAXTPU for Editing

Initializes LSE for editing. This routine allocates global data structures, initializes global variables, and calls the appropriate setup routines for each of the major components of the editor, including the Virtual File Manager, Screen Manager, and I/O subsystem.

Format

LSE$INITIALIZE *callback [,user_arg]*

Returns

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

Longword condition value. Most utility routines return a condition value in R0. Condition values that this routine can return are listed in the Condition Values Returned section.

Argument

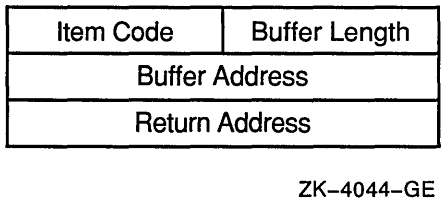
callback

VMS Usage:	**vector_longword_unsigned**
type:	**bound procedure value**
access:	**read only**
mechanism:	**by descriptor**

Callback routine. The *callback* argument is the address of a user-written routine that returns the address of an item list containing initialization parameters or a routine for handling file I/O operations. This callback routine must call a parsing routine, which can be LSE$CLIPARSE or a user-written parsing routine.

Callable LSE defines thirteen item codes that you can use for specifying initialization parameters. You do not have to arrange the item codes in any particular order in the list. Figure A–3 shows the general format of an item descriptor. For information about how to build an item list, refer to the VMS programmer's manual associated with the language you are using.

Figure A–3: Format of an Item Descriptor



The return address in an item descriptor is usually 0.

Table A–2 describes the available item codes.

Table A–2: LSE$INITIALIZE Item Codes

Item Code	Description
LSE$_OPTIONS	Enables the command qualifiers. Ten bits in the buffer-address field correspond to the various LSE command qualifiers. The remaining 22 bits in the buffer-address field are reserved.
LSE$_JOURNALFILE	Passes the string specified with the /JOURNAL qualifier. The buffer-length field is the length of the string, and the buffer-address field is the address of the string. This string is available with GET_INFO (COMMAND_LINE,"JOURNAL_FILE"). This string may be a null string.

(continued on next page)

Table A–2 (Cont.): LSE$INITIALIZE Item Codes

Item Code	Description
LSE$_SECTIONFILE	Passes the string that is the name of the binary initialization file (section file) to be mapped in. The buffer-length field is the length of the string and the buffer-address field is the address of the string. The LSE CLD file has a default value for this string. If the LSE$V_SECTION bit is set, this item code must be specified.
LSE$_OUTPUTFILE	Passes the string specified with the /OUTPUT qualifier. The buffer-length field is the length of the string, and the buffer-address field specifies the address of the string. This string is returned by the built-in procedure GET_INFO (COMMAND_LINE, "OUTPUT_FILE"). The string may be a null string.
LSE$_DISPLAYFILE	Passes the string specified with the /DISPLAY qualifier. The buffer-length field is the length of the string, and the buffer-address field specifies the address of the string.
LSE$_COMMANDFILE	Passes the string specified with the /COMMAND qualifier. The buffer-length field is the length of the string, and the buffer-address field is the address of the string. This string is returned by the built-in procedure GET_INFO (COMMAND_LINE, "COMMAND_FILE"). The string may be a null string.
LSE$_FILENAME	Passes the string that is the name of the input file specified in the command line. The buffer-length field specifies the length of this string, and the buffer-address field specifies its address. This string is returned by the built-in procedure GET_INFO (COMMAND_LINE, "FILE_NAME"). This file name may be a null string.

(continued on next page)

Table A–2 (Cont.): LSE$INITIALIZE Item Codes

Item Code	Description
LSE$_FILEIO	Passes the bound procedure value of a routine to be used for handling file operations. You may provide your own file I/O routine, or you can call LSE$FILEIO, the utility routine provided by LSE for handling file operations. The buffer-address field specifies the address of a two-longword vector. The first longword of the vector contains the address of the routine. The second longword specifies the environment value that LSE loads into R1 before calling the routine.
LSE$_CALLUSER	Passes the bound procedure value of the user-written routine that the built-in procedure CALL_USER is to call. The buffer-address field specifies the address of a two-longword vector. The first longword of the vector contains the address of the routine. The second longword specifies the environment value that LSE loads into R1 before calling the routine.
LSE$_INIT_FILE	Passes the string specified with the /INITIALIZATION qualifier. The buffer-length field is the length of the string, and the buffer-address field is the address of the string. This string is returned by using the built-in procedure GET_INFO (COMMAND_LINE,"INIT_FILE").
LSE$_START_LINE	Passes the starting line number for the edit. The buffer-address field contains the first of the two integer values you specified as part of the /START_POSITION command qualifier. The value is available by using the built-in procedure GET_INFO (COMMAND_LINE,"LINE"). Usually an initialization procedure uses this information to set the starting position in the main editing buffer. The first line in the buffer is line 1.

(continued on next page)

Table A–2 (Cont.): LSE$INITIALIZE Item Codes

Item Code	Description
LSE$_START_CHAR	Passes the starting column position for the edit. The buffer-address field contains the second of the two integer values you specified as part of the /START_POSITION command qualifier. The value is available using the built-in procedure GET_INFO (COMMAND_LINE, "CHARACTER"). Usually an initialization procedure uses this information to set the starting position in the main editing buffer. The first column on a line corresponds to character 1.
LSE$_CTRL_C_ROUTINE	Passes the bound procedure value of a routine to be used for handling CTRL/C ASTs. LSE calls the routine when a CTRL/C AST occurs. If the routine returns a FALSE value, LSE assumes that the CTRL/C has been handled. If the routine returns a TRUE value, LSE aborts any currently executing LSE procedure. The buffer-address field specifies the address of a two-longword vector. The first longword of the vector contains the address of the routine. The second longword specifies the environment value that LSE loads into R1 before calling the routine.
LSE$_DEBUGFILE	Passes the string specified with the /DEBUG command qualifier. The buffer-length field is the length of the string, and the buffer-address field is the address of the string.

Table A–3 shows the bits and corresponding masks enabled by the item code LSE$_OPTIONS.

Table A–3: LSE$_OPTIONS Masks and Bits

Mask	Bit	Function
LSE$M_RECOVER[1]	LSE$V_RECOVER[2]	Performs a recovery operation.
LSE$M_JOURNAL	LSE$V_JOURNAL	Journals the edit session.
LSE$M_READ	LSE$V_READ	Makes this a READ_ONLY edit session for the main buffer.
LSE$M_SECTION	LSE$V_SECTION	Maps in a binary initialization file (a VAXTPU section file) during startup.
LSE$M_CREATE	LSE$V_CREATE	Creates an input file if the one specified does not exist.
LSE$M_OUTPUT	LSE$V_OUTPUT	Writes the modified input file upon exiting.
LSE$M_COMMAND	LSE$V_COMMAND	Executes a command file during startup.
LSE$M_DISPLAY	LSE$V_DISPLAY	Attempts to use the terminal for screen-oriented editing and display purposes.
LSE$M_INIT	LSE$V_INIT	Indicates the presence of an initialization file.
LSE$M_COMMAND_DFLTED	LSE$V_COMMAND_DFLTED	Indicates whether the user defaulted the name of the command line. A setting of TRUE means the user did not specify a command file. If this bit is set to FALSE and the user did not specify a file, LSE$INITIALIZE fails.
LSE$M_WRITE	LSE$V_WRITE	Indicates whether the /WRITE qualifier was specified on the command line.

[1]LSE$M . . . indicates a mask.
[2]LSE$V . . . indicates a bit item.

(continued on next page)

Table A–3 (Cont.): LSE$_OPTIONS Masks and Bits

Mask	Bit	Function
LSE$M_MODIFY	LSE$V_MODIFY	Indicates whether the /MODIFY qualifier was specified on the command line.
LSE$M_NOMODIFY	LSE$V_NOMODIFY	Indicates whether the /NOMODIFY qualifier was specified on the command line.
LSE$M_DEBUG	LSE$V_DEBUG	Indicates whether the /DEBUG qualifier was specified.

To create the bits, start with the value 0, then use the OR operator on the mask (LSE$M . . .) of each item you want to set. Another way to create the bits is to treat the 32 bits as a bit vector and set the bit (LSE$V . . .) corresponding to the item you want.

user_arg

VMS Usage:	**user_arg**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by value**

User argument. The *user_arg* argument is passed to the user-written initialization routine INITIALIZE.

The *user_arg* argument is provided to allow an application to pass information through LSE$INITIALIZE to the user-written initialization routine. LSE does not interpret this data in any way.

Condition Values Returned

TPU$_SUCCESS	Initialization was completed successfully.
TPU$_SYSERROR	A system service did not work correctly.
TPU$_NONANSICRT	The input device (SYS$INPUT) is not a supported terminal.
TPU$_RESTOREFAIL	An error occurred during the restore operation.
TPU$_NOFILEROUTINE	No routine has been established to perform file operations.
TPU$_INSVIRMEM	Insufficient virtual memory exists for the editor to initialize.
TPU$_FAILURE	General code for all other errors during initialization.

Description

The LSE$INITIALIZE routine is the first routine that must be called after establishing a condition handler.

This routine initializes the editor according to the information received from the callback routine. The initialization routine defaults all file specifications to the null string and all options to off. However, it does not default the file I/O or call-user routine addresses.

If you do not specify a section file, the software features of the editor are limited.

LSE$LSE—Invoke LSE

Invokes LSE and is equivalent to the DCL command LSEDIT.

Format

LSE$LSE *command*

Returns

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

Longword condition value. Most utility routines return a condition value in R0. Condition values that this routine can return are listed in the Condition Values Returned section.

ARGUMENT

command

VMS Usage:	**char_string**
type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

Command string. The *command* argument is the address of a descriptor of a command line.

Condition Values Returned

The LSE$LSE routine returns any condition value returned by LSE$INITIALIZE, LSE$EXECUTE_INFILE, LSE$CONTROL, and LSE$CLEANUP.

Description

The LSE$LSE routine takes the command string specified and passes it to the editor. LSE uses the information from this command string for initialization purposes, just as though you had typed in the command at the DCL level.

Using the simplified callable interface does not set LSE$V_CLOSE_SECTION. This feature allows you to make multiple calls to LSE$LSE without requiring you to open and close the section file on each call.

If your application parses information that is not related to the operation of LSE, make sure the application gets, and uses, all non-LSE parse information before the application calls LSE$LSE. LSE$LSE destroys all parse information obtained and stored before LSE$LSE was called.

LSE$MESSAGE—Write Message String

Writes error messages and strings by using the built-in procedure, MESSAGE.

You can call this routine to have messages written and handled in a manner consistent with LSE. This routine should be used only after LSE$EXECUTE_INIFILE.

Format

LSE$MESSAGE *string*

Returns

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

Longword condition value.

NOTE

The return status should be ignored because it is intended for use by the Put Message (SYS$PUTMSG) system service.

Argument

string

VMS Usage:	**char_string**
type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

Formatted message. The *string* argument is the address of a descriptor of text to be written. It must be completely formatted. This routine does not append the message prefixes. However, the text is appended to the message buffer if one exists. In addition, if the buffer is mapped to a window, the window is updated.

LSE$PARSEINFO—Parse Command Line and Build Item List

Parses a command and builds the item list for LSE$INITIALIZE.

Format

LSE$PARSEINFO *fileio, call_user*

Returns

VMS Usage:	**item_list**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

The routine returns the address of an item list.

Arguments

fileio

VMS Usage:	**vector_longword_unsigned**
type:	**bound procedure value**
access:	**read only**
mechanism:	**by descriptor**

File I/O routine. The *fileio* argument is the address of a descriptor of a file I/O routine.

call_user

VMS Usage:	**vector_longword_unsigned**
type:	**bound procedure value**
access:	**read only**
mechanism:	**by descriptor**

Call-user routine. The *call_user* argument is the address of a descriptor of a call-user routine.

Description

The LSE$PARSEINFO routine parses a command and builds the item list for LSE$INITIALIZE.

This routine uses the Command Language Interpreter (CLI) routines to parse the current command. It makes queries about the command parameters and qualifiers that LSE expects. The results of these queries are used to set up the proper information in an item list. The addresses of the user routines are used for those items in the list. The address of this list is the return value of the routine.

If your application parses information that is not related to the operation of LSE, make sure the application gets, and uses, all non-LSE parse information before the application calls LSE$PARSEINFO interface. LSE$PARSEINFO destroys all parse information obtained and stored before LSE$PARSEINFO was called.

FILEIO—User-Written Routine to Perform File Operations

Handles LSE file operations. The name of this routine can be either your own file I/O routine or the name of the LSE file I/O routine (LSE$FILEIO).

Format

FILEIO *code, stream, data*

Returns

VMS Usage:	**cond_value**
type:	**longword (usigned)**
access:	**write only**
mechanism:	**by reference**

Longword condition value. Most utility routines return a condition value in R0. Condition values that this routine can return are listed in the Condition Values Returned section.

Arguments

code

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Item code specifying an LSE function. The *code* argument is the address of a longword containing an item code from LSE that specifies a function to perform.

stream

VMS Usage:	**unspecified**
type:	**longword (unsigned)**
access:	**modify**
mechanism:	**by reference**

File description. The *stream* argument is the address of a data structure containing four longwords. This data structure is used to describe the file to be manipulated.

data

VMS Usage:	**item_list_3**
type:	**longword (unsigned)**
access:	**modify**
mechanism:	**by reference**

Stream data. The *data* argument is either the address of an item list or the address of a descriptor.

NOTE

The value of this parameter depends on which item code you specify.

Condition Values Returned

The condition values returned are determined by the user and should indicate success or failure of the operation.

Description

The bound procedure value of the FILEIO routine is specified in the item list built by the callback routine. This routine is called to perform file operations. Instead of using your own file I/O routine, you can call LSE$FILEIO and pass it the parameters for any file operation that you do not want to handle. Note, however, that LSE$FILEIO must handle all I/O requests for any file it opens. Also, if it does not open the file, it cannot handle any I/O requests for the file. In other words, you cannot intermix the file operations between your own file I/O routine and the one supplied by LSE.

HANDLER—User-Written Condition Handling Routine

Performs condition handling. It is a user-written routine.

Format

HANDLER *signal_vector, mechanism_vector*

Returns

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

Longword condition value.

Arguments

signal_vector

VMS Usage:	**arg_list**
type:	**longword (unsigned)**
access:	**modify**
mechanism:	**by reference**

Signal vector. See the *VMS System Services Reference Manual* for information about the signal vector passed to a condition handler.

mechanism_vector

VMS Usage:	**arg_list**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

Mechanism vector. See the *VMS System Services Reference Manual* for information about the mechanism vector passed to a condition handler.

Description

If you need more information about writing condition handlers and the VAX Condition Handling Standard, refer to the *Introduction to VMS System Routines*.

Instead of writing your own condition handler, you can use the default condition handler, LSE$HANDLER. If you want to write your own routine, you must call LSE$HANDLER with the same parameters that your routine received to handle LSE internal signals.

INITIALIZE—User-Written Initialization Routine

Is passed to LSE$INITIALIZE as a bound procedure value and called to supply information needed to initialize LSE. It is a user-written routine.

Format

INITIALIZE *[user_arg]*

Returns

VMS Usage:	**item_list**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by reference**

This routine returns the address of an item list.

Arguments

user_arg

VMS Usage:	**user_arg**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by value**

User argument.

Description

The user written INITIALIZE routine is passed to LSE$INITIALIZE as a bound procedure value and called to supply information needed to initialize LSE.

If the *user_arg* parameter was specified in the call to LSE$INITIALIZE, the initialization call-back routine is called with only that parameter. If *user_arg* was not specified in the call to LSE$INITIALIZE, the initialization call-back routine is called with no parameters.

The *user_arg* parameter is provided to allow an application to pass information through LSE$INITIALIZE to the user-written initialization routine. LSE does not interpret this data in any way.

The user written call-back routine is expected to return the address of an item list containing initialization parameters. Because the item list is used outside the scope of the initialization call-back routine, it should be allocated in static memory.

The item list entries are discussed in the section on LSE$INITIALIZE. Most of the initialization parameters have a default value: strings default to the null string, and flags default to false. The only required initialization parameter is the address of a routine for file I/O. If an entry for the file I/O routine address is not present in the item list, LSE$INITIALIZE returns with a failure status.

USER—User-Written Routine Called from a LSE Editing Session

Allows your program to get control during an LSE editing session (for example, to leave the editor temporarily and perform a calculation).

This user-written routine is invoked by the VAXTPU built-in procedure CALL_USER. The built-in procedure CALL_USER passes three parameters to this routine. These parameters are then passed to the appropriate part of your application to be used as specified. (For example, they may be used as operands in a calculation within a FORTRAN program.) Using the string routines provided by the VMS Run-Time Library, your application fills in the *stringout* parameter in the call-user routine, which returns the *stringout* value to the built-in procedure CALL_USER.

Format

USER *integer, stringin, stringout*

Returns

VMS Usage:	**cond_value**
type:	**longword (unsigned)**
access:	**write only**
mechanism:	**by value**

Longword condition value.

Arguments

integer

VMS Usage:	**longword_unsigned**
type:	**longword (unsigned)**
access:	**read only**
mechanism:	**by descriptor**

USER

First parameter to the built-in procedure CALL_USER. This is an input-only parameter and must not be modified.

stringin

VMS Usage:	**char_string**
type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

Second parameter to the built-in procedure CALL_USER. This is an input-only parameter and must not be modified.

stringout

VMS Usage:	**char_string**
type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

Return value for the built-in procedure CALL_USER. Your program should fill in this descriptor with a dynamic string allocated by the string routines provided by the VMS Run-Time Library. LSE frees this string when necessary.

Description

The description of the built-in procedure CALL_USER in the *VAX Text Processing Utility Manual* shows an example of a BASIC program that is a call-user routine.

Example

```
INTEGER FUNCTION LSE$CALLUSER (x,y,z)
     IMPLICIT NONE
     INTEGER X
     CHARACTER*(*) Y
     STRUCTURE /dynamic/ Z
                INTEGER*2 length
                BYTE    dtype
                BYTE    class
                INTEGER ptr
     END STRUCTURE
     RECORD /dynamic/ Z
     CHARACTER*80 local_copy
     INTEGER rs,lclen
     INTEGER STR$COPY_DX
     local_copy = '<' // y // '>'
     lclen = LEN(Y) + 2

     RS = STR$COPY_DX(Z,local_copy(1:lclen))
     LSE$CALLUSER = RS
     END
```

You can call this FORTRAN program with a VAXTPU procedure. The following is an example of one such procedure:

```
PROCEDURE MY_CALL
     local status;
     status := CALL_USER (0,'ABCD');
     MESSAGE('"' + '"');
     ENDPROCEDURE
```

Appendix B
VAX Source Code Analyzer (SCA) Callable Interface

With the VAX Source Code Analyzer callable interface, you can use SCA within independent application programs. You can integrate SCA into alternative user-interfaces and generate specialized reports based on SCA information.

The SCA callable interface provides a set of callable routines. You should have an understanding of basic SCA concepts before using these routines.

This appendix contains the following information:

- An overview of the SCA callable interface
- Rules for calling SCA routines
- Rules for Calling SCA routines from LSE/VAXTPU
- A description of message handling
- A description of the callable command interface
- A description of the callable query interface
- SCA routine descriptions

B.1 Overview

The SCA callable interface contains two components. The first is a set of callable command routines. The callable command routines comprise a high-level interface that you must always use, regardless of the type of application. This provides a very simple callable interface to SCA that is sufficient for most applications.

The second component is a set of callable query routines. The callable query routines comprise a lower-level interface to the FIND command. When you use this interface, your application has control over the specification of queries and the manipulation of query results.

B.2 Message Handling

The SCA callable interface handles all messages the same way: it signals them.

If you want control over the display of such messages, you must establish a condition handler. Establishing a condition handler is optional.

B.3 Rules for Calling SCA Routines

The following list describes rules for programs that call SCA routines.

- You must call SCA$CLEANUP to terminate callable SCA.
- Most SCA routines are not AST-reentrant; therefore, you should not call an SCA routine (except SCA$ASYNCH_TERMINATE) from an AST routine that may currently be interrupting an SCA routine.
- Your program must not disable ASTs.
- If your program uses event flags, you must use the VMS RTL routines (LIB$RESERVE_EF, LIB$GET_EF, and LIB$FREE_EF) to coordinate the use of event flags between your program and SCA.
- Except for SCA$ASYNCH_TERMINATE, do not call SCA from within an SCA callback routine or from within a routine that is handling a condition signaled by SCA.
- Your program must not unwind when handling a condition signaled by SCA.

B.4 Rules for Calling SCA Routines from LSE/VAXTPU

A subset of the SCA callable interface routines may be called from VAXTPU code executed from within the VAX Language-Sensitive Editor. The following list describes rules for calling SCA routines from VAXTPU:

- You must use LSE to execute your VAXTPU code. The SCA routines are not available from the version of VAXTPU provided with VMS.

- LSE uses the VAXTPU variable LSE$SCA_COMMAND_CONTEXT to store the command context that is created when initializing SCA. When you call the SCA routine SCA$QUERY_INITIALIZE, you must pass this variable as the command context parameter. You should not change this variable.
- Only the following subset of routines are available. In particular, note that you cannot call SCA$INITIALIZE or SCA$CLEANUP from VAXTPU. LSE will handle the calls to the following routines for you, automatically:
  - SCA$QUERY_CLEANUP
  - SCA$QUERY_COPY
  - SCA$QUERY_FIND
  - SCA$QUERY_GET_ATTRIBUTE
  - SCA$QUERY_GET_ATTRI_KIND_T
  - SCA$QUERY_GET_ATTRI_VALUE_T
  - SCA$QUERY_GET_DESCRIPTION
  - SCA$QUERY_GET_OCCURRENCE
  - SCA$QUERY_GET_NAME
  - SCA$QUERY_INITIALIZE
  - SCA$QUERY_PARSE
  - SCA$QUERY_SELECT_OCCURRENCE
- SCA query contexts and entity handles are represented as VAXTPU integers.
- SCA attribute constants, of the form SCA$K_ATTRI_attribute_name, are defined as TPU constants in the file SYS$LIBRARY:SCA$QUERY_CALLABLE.TPU. You must read and execute this file before using any of the SCA$K_ATTRI_attribute_name constants.

B.5 Callable Command Interface

The callable command routines comprise the high-level interface that you must always use, regardless of the type of application. This provides a simple callable interface to SCA that is sufficient for most applications.

You use the SCA$INITIALIZE routine to initialize SCA. You must call this routine before any other SCA routine. SCA$INITIALIZE creates a **command-context**. A command-context is represented as a longword-sized value that SCA$INITIALIZE returns to the application. This command-context value must be passed to all of the other command routines, and to the SCA$QUERY_INITIALIZE routine.

You use the SCA$CLEANUP routine to terminate a command-context. It frees the SCA internal data structures and severs all connections to SCA libraries and servers.

Typically, an application has only one command-context at a time. However, you can call SCA$INITIALIZE more than once without calling SCA$CLEANUP. Each call to SCA$INITIALIZE creates a separate command-context. Each separate command-context behaves independently from all other command-contexts. With more than one command-context, an application can query two different virtual libraries at the same time without having to reset the virtual library. Note that process quotas may place limits on the number of command-contexts you can run simultaneously.

You use the SCA$DO_COMMAND routine to execute SCA commands. For all commands except FIND, SCA$DO_COMMAND is the only available routine for executing an SCA command. It gives the application control over the user interface of any SCA command.

In addition, there are two routines, SCA$LOCK_LIBRARY and SCA$UNLOCK_LIBRARY, for creating and deleting write-locks on current virtual library lists. With the SCA$LOCK_LIBRARY routine, you can lock the virtual library so that it cannot be modified between successive calls to the SCA$DO_COMMAND or SCA$QUERY_FIND routines.

B.5.1 Example of the SCA Callable Interface

The following is an example of an application routine using the SCA callable command interface. This example shows a simple routine that automatically loads .ANA files following compilations.

```
ROUTINE load_ana_file( ana_file_spec : _string ) =
!++
! FUNCTIONAL DESCRIPTION:
!
!   LOAD_ANA_FILE loads the .ANA file specified as ANA_FILE_SPEC
!   into the current SCA library.
!
! MACROS:
!
!   _COPY_STRING concatenates several strings into a single
!       destination string.  It is similar to STR$CONCAT.
!   _DYNAMIC_STRING declares and initializes a dynamic string descriptor.
!   _FREE_STRING releases the memory described by a dynamic string
!        descriptor.
!   _STRING declares a string descriptor, but does not initialize it.
!--
    BEGIN
    LOCAL
        cmdline : _dynamic_string
        sca_context;

    ! Initialize SCA.
    !
    sca$initialize( sca_context );

    ! Load the .ANA file.
    !
    _copy_string( cmdline, 'LOAD ', .ana_file_spec )
    sca$do_command( sca_context, cmdline );

    ! Free the memory associated with the dynamic string descriptor.
    !
    _free_string( cmdline );

    ! Cleanup SCA.
    !
    sca$cleanup( sca_context )
    END;
```

B.6 Callable Query Interface

The callable query routines comprise the second component of the SCA callable interface. This component is a lower-level interface to the FIND command. Using this interface, an application has control over the specification of queries and the manipulation of query results.

As with the callable command interface, you begin using the callable query interface by issuing an SCA$QUERY_INITIALIZE routine. However, instead of creating a command-context, this routine initializes a **query-context**. As with the callable command interface, SCA$QUERY_INITIALIZE must be called before any other query routine. A query-context is represented as a longword-sized value that SCA$QUERY_INITIALIZE returns to the application. This query-context value must be passed to many of the other query routines.

Similarly, you use the SCA$QUERY_CLEANUP routine to end a query-context using SCA. It frees the internal data structures that represent the query.

Typically, an application uses several query-contexts at the same time. Each call to SCA$QUERY_INITIALIZE creates a separate query-context.

In SCA, the term *query* refers to both a question and its corresponding result. There is a one-to-one correspondence between a query and a query-context. A query-context first expresses a question. The SCA$QUERY_FIND routine then evaluates the query. This creates a **query result**. Finally, the result is analyzed. You may then issue the SCA$QUERY_SELECT_OCCURRENCE routine to use the results of one query in forming the question of another query.

Definitions required to use the SCA callable query interface are stored in files called SYS$LIBRARY:SCA$QUERY_CALLABLE.*. Each language has its own definition file. For example, the Pascal definition file is called SYS$LIBRARY:SCA$QUERY_CALLABLE.PAS.

B.6.1 Data Models

The callable query interface uses an entity-relationship-attribute data model. Entities are atomic elements of the database. Two entities can have a relationship. A relationship has associated with it two entities: one is the source, the other is the target. Both entities and relationships have attributes.

In the SCA callable query interface, the only kind of entity that is presented is **occurrence**. These occurrences have many attributes, for example, an occurrence can have a **name**.

B.6.2 Handles

The callable query interface uses **handles** to refer to entities and attributes. Handles are generic. You can use a particular handle to refer to an entity or an attribute, or with any query-context.

A handle is a longword-value. Before using a handle for the first time, it must be set to zero. Thereafter you can use it in a routine that requires a handle as an output parameter, such as SCA$QUERY_GET_OCCURRENCE. After that you can use it either as an output parameter or as an input parameter.

Once you use a handle to refer to something, it becomes dependent on the query-context that contains its referrant. If a query-context is deleted, any handles associated with that query-context become invalid. An invalid handle must be reinitialized before you use it again; that is, it must be set to zero. Any use of a query-context as an output parameter also invalidates any associated handles. This includes any question-building routine, such as SCA$QUERY_PARSE and SCA$QUERY_FIND.

SCA attempts to detect the use of an invalid handle and signal an error message. Such use is considered a programmer error on the part of the application.

B.6.3 Entities

Entities are atomic elements of the database and have attributes.

An entity is referred to with a handle. A handle that refers to an entity is called an **entity handle**. You can use the SCA$QUERY_GET_OCCURRENCE routine to visit all of the entities in a particular query result.

B.6.4 Attributes

Entities have attributes, that provide information about the entity. You use attributes to do different things, including express questions and retrieve results.

The SCA callable query interface supports a number of different kinds of attributes. Each **attribute-kind** has associated with it an attribute-value of a particular data-type. All attribute-values are returned as character strings.

The following is a list of the attribute-kinds. They are described in more detail later.

- SCA$K_ATTRI_APPEARANCE
- SCA$K_ATTRI_DOMAIN
- SCA$K_ATTRI_EXPRESSION
- SCA$K_ATTRI_LANGUAGE
- SCA$K_ATTRI_NAME
- SCA$K_ATTRI_OCCURRENCE_CLASS
- SCA$K_ATTRI_SYMBOL_CLASS

- SCA$K_ATTRI_MACH_DSIZE
- SCA$K_ATTRI_MACH_DTYPE
- SCA$K_ATTRI_PASSING_MECHANISM
- SCA$K_ATTRI_FILE_SPEC
- SCA$K_ATTRI_BEGIN_RECORD_NUMBER
- SCA$K_ATTRI_NAME_RECORD_NUMBER
- SCA$K_ATTRI_BEXE_RECORD_NUMBER
- SCA$K_ATTRI_END_RECORD_NUMBER
- SCA$K_ATTRI_BEGIN_CHAR_OFFSET
- SCA$K_ATTRI_NAME_CHAR_OFFSET
- SCA$K_ATTRI_BEXE_CHAR_OFFSET
- SCA$K_ATTRI_END_CHAR_OFFSET
- SCA$K_ATTRI_ALL

You can use the SCA$K_ATTRI_ALL attribute-kind in calls to SCA$QUERY_GET_ATTRIBUTE to find all attributes, regardless of kind.

Some attribute-kinds correspond to attribute-names in the Query Language that have similar names. For example, the SCA$K_ATTRI_NAME attribute-kind corresponds to the NAME attribute-name. Other attribute-kinds have a different representation at the command line, and some attribute-kinds, such as SCA$K_ATTRI_PASSING_MECHANISM, have no analog in the Query Language.

B.6.4.1 Attribute Retrieval

Attribute-retrieval is the process of getting attributes of entities in a query result. Attributes are retrieved using the SCA$QUERY_GET_ATTRIBUTE and SCA$QUERY_GET_ATTRI_VALUE_T routines.

There are two ways to use these routines. The first way is to use the get-attribute-value routine to find the value of a particular attribute-kind for an entity. For example, if OCC is a handle to an occurrence whose name you want to retrieve into the string-descriptor NAME, you use the following call:

```
sca$query_get_attri_value_t( occ, name, %REF(sca$k_attri_name) );
```

The string-descriptor NAME now contains the name of the occurrence, for example, XYZ.

You retrieve the value of any attribute as a character-string. For example, you can use the following call to retrieve a string representation of the symbol-class in the string-descriptor SYMBOL_CLASS.

```
sca$query_get_attri_value_t( occ,
                             symbol_class,
                             %REF(sca$k_attri_symbol_class) );
```

The string-descriptor SYMBOL_CLASS now describes the character-string VARIABLE.

The second way is to use the SCA$QUERY_GET_ATTRIBUTE routine to get more than one attribute for an entity. This routine returns a handle to an attribute. You can then use that handle in calls to SCA$QUERY_GET_ATTRI_VALUE_T to get the value of the attribute.

Also, with the SCA$QUERY_GET_ATTRI_KIND_T routine, you can get the *kind* of the attribute. The following example illustrates the use of this routine and highlights the manipulation of attributes.

```
ROUTINE dump_entity( entity ) =
!++
! FUNCTIONAL DESCRIPTION
!
!       Display all of the attributes of an entity.
!
! FORMAL PARAMETERS:
!
!   entity
!       The address of an SCA handle describing the entities whose
!       attributes are to·be displayed.
!
! MACROS:
!
!   _COPY_STRING concatenates several strings into a single
!       destination string.  It is similar to STR$CONCAT.
!   _DYNAMIC_STRING declares and initializes a dynamic string descriptor.
!   _FREE_STRING releases the memory described by a dynamic string
!        descriptor.
!--
    BEGIN
    LOCAL
        attribute : INITIAL(0),             ! handle for attributes
        display_line : _dynamic_string,     ! one line of display output
        iteration_ctx : INITIAL(0),         ! iteration-context
        kind : _dynamic_string,             ! attribute-kind as a string
        value: _dynamic_string;             ! attribute-value as a string
```

```
! Loop once for every attribute of the specified entity.
!
WHILE sca$query_get_attribute( .entity,
                               %REF(sca$k_attri_all),
                               attribute,
                               iteration_ctx ) DO
    BEGIN
    ! Get the attribute-kind and the attribute-value.
    !
    sca$query_get_attri_kind_t( attribute, kind );
    sca$query_get_attri_value_t( attribute, value );

    ! Create a line of output (e.g., "NAME=FOO") and write it out.
    !
    _copy_string( display_line, kind, '=', value );
    lib$put_output( display_line );
    END;

_free_string( display_line, kind, value )
END;
```

B.6.4.2 Appearance

You specify the *occurrence-appearance* attribute with the SCA$K_ATTRI_APPEARANCE attribute-kind. *Occurrence-appearance* indicates that the name of the symbol actually appears in the source code, or that it was hidden.

The *occurrence-appearance* attribute corresponds to portions of the *occurrence-class* attribute in the Query Language. In particular, it corresponds to the attribute-selection expressions occ=visible and occ=hidden.

B.6.4.3 Symbol Domain

You specify the *symbol-domain* attribute with the SCA$K_ATTRI_DOMAIN attribute-kind. *Symbol-domain* indicates the range of source code in which a symbol has the potential of being used.

The *symbol-domain* attribute corresponds to the domain=symbol-domain-value attribute-selection expression in the Query Language.

B.6.4.4 Expression

You specify the *occurrence-expression* attribute with the SCA$K_ATTRI_EXPRESSION attribute-kind. *Occurrence-expression* indicates that the occurrence explicitly appears in the source file, or that the occurrence was implicit. For example, in FORTRAN, you can implicitly declare a variable. As a FORTRAN programmer, you might think that the variable is not declared. The SCA view is that it is declared, but it is declared implicitly.

The *occurrence-expression* attribute corresponds to the portions of the *occurrence-class* attribute in the Query Language. In particular, it corresponds to the attribute-selection expressions occ=explicit and occ=implicit.

B.6.4.5 Language

You specify the *language* attribute with the SCA$K_ATTRI_LANGUAGE attribute-kind. *Language* indicates the programming language of the occurrence.

The *language* does not correspond to anything in the Query Language.

B.6.4.6 Name

You specify the *name* attribute with the SCA$K_ATTRI_NAME attribute-kind. *Name* indicates the name of the symbol.

B.6.4.7 Occurrence Class

You specify the *occurrence-class* attribute with the SCA$K_ATTRI_OCCURRENCE_CLASS attribute-kind. *Occurrence-class* indicates the kind of occurrence.

The *occurrence-class* attribute corresponds to the occurrence=occurrence-class-value attribute-selection expression.

B.6.4.8 Symbol Class

You specify the *symbol-class* attribute with the SCA$K_ATTRI_SYMBOL_CLASS attribute-kind. *Symbol-class* indicates the kind of symbol.

The *symbol-class* attribute corresponds to the symbol=symbol-class-value attribute-selection expression.

B.6.4.9 Passing Mechanism

You specify the *passing-mechanism* attribute with the SCA$K_ATTRI_PASSING_MECHANISM attribute-kind. *Passing-mechanism* applies only to argument declarations. It indicates the mechanism by which an argument is passed to a routine.

The *passing-mechanism* attribute does not correspond to anything in the Query Language.

B.6.4.10 File Specification

You specify the *file-specification* attribute with the SCA$K_ATTRI_FILE_SPEC attribute-kind. *File-specification* describes the file specification of the source file that contains the occurrence.

The *file-specification* attribute does not correspond to anything in the Query Language.

B.6.4.11 Begin Record Number

You specify the *begin-record-number* attribute with the SCA$K_ATTRI_BEGIN_RECORD_NUMBER attribute-kind. *Begin-record-number* describes the source-file record number of the beginning of the lexical range of a declaration. It is not applicable to references, which are assumed to have a single lexical location.

The *begin-record-number* attribute does not correspond to anything in the Query Language.

B.6.4.12 Name Record Number

You specify the *name-record-number* attribute with the SCA$K_ATTRI_NAME_RECORD_NUMBER attribute-kind. *Name-record-number* describes the source-file record number where the name of the symbol appears.

The *name-record-number* attribute does not correspond to anything in the Query Language.

B.6.4.13 Begin Executable Record Number

You specify the *begin-executable-record-number* attribute with the SCA$K_ATTRI_BEXE_RECORD_NUMBER attribute-kind. *Begin-executable-record-number* describes the source-file record number where the executable part of a routine begins. This attribute is primarily applicable to primary routine declarations.

The *begin-executable-record-number* attribute does not correspond to anything in the Query Language.

B.6.4.14 End Record Number

You specify the *end-record-number* attribute with the SCA$K_ATTRI_END_RECORD_NUMBER attribute-kind. *End-record-number* describes the source-file record number of the end of the lexical range of a declaration. It is not applicable to references, which are assumed to have a single lexical location.

The *end-record-number* attribute does not correspond to anything in the Query Language.

B.6.4.15 Begin Character Offset

You specify the *begin-character-offset* attribute with the SCA$K_ATTRI_BEGIN_CHAR_OFFSET attribute-kind. *Begin-character-offset* describes the source-file character offset of the beginning of the lexical range of a declaration. It is not applicable to references, which are assumed to have a single lexical location.

The *begin-character-offset* attribute does not correspond to anything in the Query Language.

B.6.4.16 Name Character Offset

You specify the *name-character-offset* attribute with the SCA$K_ATTRI_NAME_CHAR_OFFSET attribute-kind. *Name-character-offset* describes the source-file character offset where the name of the symbol appears.

The *name-character-offset* attribute does not correspond to anything in the Query Language.

B.6.4.17 Begin Executable Character Offset

You specify the *begin-executable-character-offset* attribute with the SCA$K_ATTRI_BEXE_CHARACTER_OFFSET attribute-kind. *Begin-executable-character-offset* describes the source-file character offset where the executable part of a routine begins. This attribute is primarily applicable to primary routine declarations.

The *begin-executable-character-offset* attribute does not correspond to anything in the Query Language.

B.6.4.18 End Character Offset

You specify the *end-character-offset* attribute with the SCA$K_ATTRI_END_CHAR_OFFSET attribute-kind. *End-character-offset* describes the source-file character offset of the end of the lexical range of a declaration. It is not applicable to references, which are assumed to have a single lexical location.

The *end-character-offset* attribute does not correspond to anything in the Query Language.

B.6.5 Example of the Callable Query Interface

The following is an example of an application using the callable query interface. It shows the building of questions, their evaluation, and the use of the query results.

```
ROUTINE display_tags_of_decl( sca_cmd_ctx, decl_name : _string ) =
!++
! FUNCTIONAL DESCRIPTION
!
!       Display info about all of the tags associated with primary
!       declarations with the specified name.
!
! FORMAL PARAMETERS:
!
!   sca_cmd_ctx
!            An SCA command context.  It has already been initialized.
!            The SCA library has already been set.
!
!   decl_name
!            Name of the declaration.
!
! MACROS:
!
!   _COPY_STRING concatenates several strings into a single
!       destination string.  It is similar to STR$CONCAT.
!   _DYNAMIC_STRING declares and initializes a dynamic string descriptor.
!   _FREE_STRING releases the memory described by a dynamic string
!        descriptor.
!   _STRING declares a string descriptor, but does not initialize it.
!--
    BEGIN
    LOCAL
        all_tags_query,              ! query context for "SYMBOL=TAG"
        cmdline : _dynamic_string,   ! temp for holding FIND command line
        decl : INITIAL(0),           ! handle for the named declarations
        decl_query,                  ! query context for the named decls
        particular_decl_query,       ! query context for particular decl
        tag : INITIAL(0),            ! handle for tag declarations
        tag_name : _dynamic_string,  ! holds the tag name strings
        tag_query;                   ! query context for tags

    ! Initialize the necessary query contexts.
    !
    sca$query_initialize( .sca_cmd_ctx, all_tags_query );
    sca$query_initialize( .sca_cmd_ctx, decl_query );
    sca$query_initialize( .sca_cmd_ctx, particular_decl_query );
    sca$query_initialize( .sca_cmd_ctx, tag_query );

    ! Lock the virtual library so that it cannot change out from under
    ! us.  If this is not done, there is a chance that the call to
    ! SCA$QUERY_SELECT_OCCURRENCE will fail, since the specified
    ! occurrence is made obsolete if its module is updated.
    !
    sca$lock_library( .sca_cmd_ctx );

    ! Form the question: all of the primary declarations with the
    ! specified name.
    !
    _copy_string( cmdline, 'occ=primary AND ', .decl_name );
    sca$query_parse( decl_query, cmdline );
    _free_string( cmdline );

    ! Evaluate that question.
    !
    sca$query_find( decl_query );
```

```
! Form the question: SYMBOL=TAG.
!
sca$query_parse( all_tags_query, _ascid( 'symbol=tag' ) );

! Now we visit each of the primary declarations describe by DECL_QUERY.
!
WHILE sca$query_get_occurrence( decl_query, decl ) DO
    BEGIN
    ! So we have our hands on a named-declaration.  Now we want
    ! to find its corresponding tag declarations.  Begin by
    ! creating a query question that describes the name-decl.
    !
    sca$query_select_occurrence( particular_decl_query, decl );

    ! Then we get the name of that query
    !
    sca$query_get_name( particular_decl_query,
                        particular_decl_query_name );

    ! Form the question:
    !   CONTAINING( @name-of-decl-query, SYMBOL=TAG, RESULT=BEGIN )
    !
    _copy_string(
        cmdline,
        'CONTAINING( BEGIN=SYMBOL=TAG, RESULT=BEGIN, END=@',
        .particular_decl_query_name );

    sca$query_parse( tag_query, cmdline );

    ! Evaluate that query.
    !
    sca$query_find( tag_query );

    ! Visit each of the tags
    !
    WHILE sca$query_get_occurrence( tag_query, tag ) DO
        BEGIN
        ! Now we've finally got our hands on a tag declaration,
        ! let's check out its name.
        !
        sca$query_get_attri_value_t( tag,
                                     tag_name,
                                     %REF(sca$k_attri_name) );
        !
        ! TAG_NAME is now a string descriptor describing the
        ! name of the tag.

        [~etcetera~]
        END
    END;

_free_string( cmdline, tag_name, tag_query_name );

sca$unlock_library( .sca_cmd_ctx );

sca$query_cleanup( .sca_cmd_ctx, all_tags_query );
sca$query_cleanup( .sca_cmd_ctx, decl_query );
sca$query_cleanup( .sca_cmd_ctx, particular_decl_query );
sca$query_cleanup( .sca_cmd_ctx, tag_query );
END;
```

B.7 Callable SCA Routines

The callable SCA routines are described in this appendix. The callable SCA routines are divided into the following categories. The routines are described in detail later.

- Callable command interface routines
- Callable query initialization/cleanup routines
- Callable query question-building routines
- Callable query result manipulation routines
- Callable query miscellaneous routines

B.7.1 Callable Command Interface Routines

The following are callable command interface routines:

- SCA$ASYNCH_TERMINATE
- SCA$CLEANUP
- SCA$DO_COMMAND
- SCA$GET_INPUT
- SCA$INITIALIZE
- SCA$LOCK_LIBRARY
- SCA$PUT_OUTPUT
- SCA$UNLOCK_LIBRARY

B.7.2 Callable Query Initialization/Cleanup Routines

The following are callable query initialization/cleanup routines:

- SCA$QUERY_CLEANUP
- SCA$QUERY_INITIALIZE

B.7.3 Callable Query Question-Building Routines

The following are callable query question-building routines:

- SCA$QUERY_PARSE
- SCA$QUERY_SELECT_OCCURRENCE

B.7.4 Callable Query Result Manipulation Routines

The following are callable query result manipulation routines:

- SCA$QUERY_GET_ATTRIBUTE
- SCA$QUERY_GET_ATTRI_KIND_T
- SCA$QUERY_GET_ATTRI_VALUE_T
- SCA$QUERY_GET_OCCURRENCE

B.7.5 Callable Query Miscellaneous Routines

The following are callable query miscellaneous routines:

- SCA$QUERY_COPY
- SCA$QUERY_FIND
- SCA$QUERY_GET_NAME

SCA$ASYNCH_TERMINATE

Sets a flag indicating that a CTRL-C has been issued.

Format

SCA$ASYNCH_TERMINATE *command_context*

Argument

command_context

type:	**longword**
access:	**read/write**
mechanism:	**by reference**

SCA command context value.

Condition Value Returned

SCA$_NORMAL	Normal successful completion.

Description

The SCA$ASYNCH_TERMINATE routine sets a flag indicating that a CTRL-C has been issued.

SCA$CLEANUP

Shuts down the SCA callable command interface, freeing all dynamic memory associated with the interface routines and data structures.

Format

SCA$CLEANUP *command_context*

Argument

command_context

type:	**longword**
access:	**read/write**
mechanism:	**by reference**

An SCA command context value.

Condition Value Returned

SCA$_NORMAL	The SCA callable command interface has been successfully shut down.

Description

The SCA$CLEANUP routine shuts down the SCA callable command interface, freeing all dynamic memory associated with the interface routines and data structures.

SCA$DO_COMMAND

Parses an SCA subsystem command and invokes command processing if the command is syntactically correct.

Format

SCA$DO_COMMAND *command_context, command_string [,parameter_routine] [,continuation_routine] [,continuation_prompt] [,user_argument] [,confirm_routine] [,topic_routine] [,display_routine]*

Arguments

command_context

type:	**longword**
access:	**read/write**
mechanism:	**by reference**

SCA command context value.

command_string

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

An SCA subsystem command.

parameter_routine

type:	**procedure**
access:	**read only**
mechanism:	**by reference**

Routine that prompts for required parameters. You can specify LIB$GET_INPUT or a compatible routine. If a routine address of zero (0) is specified, commands with missing parameters fail and display a CLI error message.

continuation_routine

type:	**procedure**
access:	**read only**
mechanism:	**by reference**

Routine that prompts for the remainder of a continued command (i.e., a command that ends with a hyphen). You can specify LIB$GET_INPUT or a compatible routine.

continuation_prompt

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

Command continuation prompt string (e.g., SCA>).

user_argument

type:	**longword**
access:	**read only**
mechanism:	**by reference**

User-specified value to be passed to any action routine (other than CLI prompt routines) called by this routine.

confirm_routine

type:	**procedure**
access:	**read only**
mechanism:	**by value**

Command confirmation prompt routine to be used by commands that support a /CONFIRM qualifier. You can specify SCA$GET_INPUT or a compatible routine. If this argument is omitted, the /CONFIRM qualifier is not supported.

topic_routine

type:	**procedure**
access:	**read only**
mechanism:	**by value**

Help topic prompt routine. You can specify LIB$GET_INPUT or a compatible routine. If this routine returns an error, command processing is terminated. If this argument is omitted, no help prompting is performed.

display_routine

type:	**procedure**
access:	**read only**
mechanism:	**by value**

Routine to be called to display one line of command output. You can specify SCA$PUT_OUTPUT or a compatible routine. If this routine returns an error, command processing is terminated. If this argument is omitted, no display routine is called.

Condition Values Returned

All SCA condition values and
many system values.

Description

The SCA$DO_COMMAND routine parses an SCA subsystem command and invokes command processing if the command is syntactically correct.

SCA$GET_INPUT

Gets one record of ASCII text from the current controlling input device specified by SYS$INPUT.

Format

SCA$GET_INPUT *get_string,*
[,prompt_string]
[,output_length]
[,user_argument]

Arguments

get_string

type:	**character string**
access:	**write only**
mechanism:	**by descriptor**

Buffer to receive the line read from SYS$INPUT. The string is returned by a call to STR$COPY_DX.

prompt_string

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

Prompt message that is displayed on the controlling terminal. A valid prompt consists of text followed by a colon (:), a space, and no carriage-return/line-feed combination. The maximum size of the prompt message is 255 characters. If the controlling input device is not a terminal, this argument is ignored.

output_length

type:	**word**
access:	**read only**
mechanism:	**by reference**

Word to receive the actual length of the GET-STRING line, not counting any padding in the case of a fixed string. If the input line was truncated, this length reflects the truncated string.

user_argument

type:	**longword**
access:	**read only**
mechanism:	**by reference**

User-specified value that was passed to the routine that called this action routine.

Condition Values Returned

SCA$_NORMAL	An input line was returned.
Failure completion code from LIB$GET_INPUT.	

Description

The SCA$GET_INPUT routine gets one record of ASCII text from the current controlling input device specified by SYS$INPUT.

SCA$INITIALIZE

Initializes the SCA callable command interface.

Format

SCA$INITIALIZE *command_context*

Argument

command_context

type:	**longword**
access:	**write only**
mechanism:	**by reference**

SCA command context value (longword) to be initialized. This value is passed as an argument to other SCA$xxx routines.

Condition Value Returned

SCA$_NORMAL	The SCA callable command interface has been successfully initialized.

Description

The SCA$INITIALIZE routine initializes the SCA callable command interface.

SCA$LOCK_LIBRARY

Locks all the physical libraries in the current virtual library list so that they cannot be modified.

Format

SCA$LOCK_LIBRARY *command_context*

Argument

command_context

type:	**longword**
access:	**read/write**
mechanism:	**by reference**

An SCA command context.

Condition Value Returned

SCA$_NORMAL	The libraries have been successfully locked.

Description

The SCA$LOCK_LIBRARY routine locks all the physical libraries in the current virtual library list so that they cannot be modified.

SCA$PUT_OUTPUT

Writes a record to the current controlling output device specified by SYS$OUTPUT.

Format

SCA$PUT_OUTPUT *string,*
user_argument

Arguments

string

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

String to be written to SYS$OUTPUT. You can concatenate one or more additional character strings with the primary string to form a single output record. You can specify a maximum of 20 strings. The maximum resulting record length is 255 characters.

user_argument

type:	**_UNSPECIFIED**
access:	**read only**
mechanism:	**by reference**

User-specified value that was passed to the routine that called this action routine.

Condition Values Returned

SCA$_NORMAL	The string was successfully written to SYS$OUTPUT.
Failure completion code from the VAX RMS $PUT service.	

Description

The SCA$PUT_OUTPUT routine writes a record to the current controlling output device specified by SYS$OUTPUT.

SCA$UNLOCK_LIBRARY

Unlocks all the physical libraries in the current virtual library list so that they can be modified.

Format

SCA$UNLOCK_LIBRARY *command_context*

Argument

command_context

type:	**longword**
access:	**read/write**
mechanism:	**by reference**

An SCA command context.

Condition Value Returned

SCA$_NORMAL	The libraries have been successfully unlocked.

Description

The SCA$UNLOCK_LIBRARY routine unlocks all the physical libraries in the current virtual library list so that they can be modified.

SCA$QUERY_CLEANUP

Cleans up an SCA query context, freeing all dynamic memory associated with the query.

Format

SCA$QUERY_CLEANUP *query_context*

Argument

query_context

type:	**longword**
access:	**read/write**
mechanism:	**by reference**

An SCA query context to be cleaned up.

Condition Value Returned

SCA$_NORMAL	The query context has been successfully cleaned up.

Description

The SCA$QUERY_CLEANUP routine cleans up an SCA query context, freeing all dynamic memory associated with the query.

SCA$QUERY_COPY

Copies a query from SRC_QUERY_CONTEXT to DST_QUERY_CONTEXT.

Format

SCA$QUERY_COPY *src_query_context, dst_query_context*

Arguments

src_query_context

type:	**longword**
access:	**read/write**
mechanism:	**by reference**

An SCA query context that describes the query to be copied.

dst_query_context

type:	**longword**
access:	**read/write**
mechanism:	**by reference**

An SCA query context into which the query is to be copied.

Condition Value Returned

SCA$_NORMAL	The query expression has been successfully copied.

Description

The SCA$QUERY_COPY routine copies a query from SRC_QUERY_CONTEXT to DST_QUERY_CONTEXT. This will copy whatever is in SRC_QUERY_CONTEXT, whether that is a question, or a question and a result.

SCA$QUERY_FIND

Finds the occurrences that match the query expression specified by QUERY_CONTEXT.

Format

SCA$QUERY_FIND *query_context*

Argument

query_context

type:	**longword**
access:	**read/write**
mechanism:	**by reference**

An SCA query context that describes a query expression to be evaluated.

Condition Values Returned

SCA$_NORMAL	The query expression has been successfully evaluated.
SCA$_NOOCCUR	No occurrences match the query expression.
SCA$_RESULTEXISTS	The query already has a result prior to this call.

Description

The SCA$QUERY_FIND routine finds the occurrences that match the query expression specified by QUERY_CONTEXT.

SCA$QUERY_GET_ATTRIBUTE

Gets a handle to an attribute of an entity.

Format

SCA$QUERY_GET_ATTRIBUTE *entity, attribute_kind, attribute_handle [,iteration_context]*

Arguments

entity

type:	**$SCA_HANDLE_IN**
access:	**read only**
mechanism:	**by reference**

An SCA entity handle describing the entity or relationship whose attributes are being obtained.

attribute_kind

type:	**$SCA_ATTRIBUTE_KIND**
access:	**read only**
mechanism:	**by reference**

The kind of attribute to be obtained.

Any attribute-kind can be specified on this routine.

attribute_handle

type:	**$SCA_HANDLE**
access:	**write only**
mechanism:	**by reference**

An SCA attribute handle that is to describe the obtained attribute.

iteration_context

type:	**$SCA_ITERATION_CONTEXT**
access:	**read/write**
mechanism:	**by reference**

Optional. The iteration-context. This longword must contain zero on the first call to this routine for a particular iteration. This routine uses the longword to maintain the iteration context. The caller must not change the contents of the longword.

Condition Values Returned

SCA$_NORMAL	An attribute has been successfully returned.
SCA$_NONE	Warning. An attribute has not been returned. Either there are no such attributes at all in the entity or there are no more attributes.

Description

The SCA$QUERY_GET_ATTRIBUTE routine gets a handle to an attribute of an entity.

If the *iteration_context* parameter is not specified, then this routine finds the first attribute of the specified kind *(attribute_kind)* and updates *attribute_handle* to describe that attribute.

In general, several attributes can be associated with a particular entity. With this routine you can find all of those attributes by using the *iteration_context* parameter.

SCA$QUERY_GET_ATTRI_KIND_T

Gets an attribute kind.

Format

SCA$QUERY_GET_ATTRI_KIND_T *attribute_handle, attribute_kind*

Arguments

attribute_handle

type:	**$SCA_HANDLE_IN**
access:	**write only**
mechanism:	**by reference**

An SCA handle describing an attribute whose attribute-kind is to be obtained.

attribute_kind

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

The kind of the attribute.

Condition Value Returned

SCA$_NORMAL	An attribute kind has been successfully returned.

Description

The SCA$QUERY_GET_ATTRI_KIND_T routine returns the kind of any attribute as a character string.

SCA$QUERY_GET_ATTRI_VALUE_T

Gets an attribute value.

Format

SCA$QUERY_GET_ATTRI_VALUE_T *handle, attribute_value ,attribute_kind*

Arguments

handle

type:	**$SCA_HANDLE**
access:	**read/write**
mechanism:	**by reference**

An SCA attribute handle describing either an attribute or an entity whose value is to be obtained.

attribute_value

type:	**character string**
access:	**read/write**
mechanism:	**by descriptor**

The (string) value of the attribute being selected.

attribute_kind

type:	**$SCA_ATTRIBUTE_KIND**
access:	**read/write**
mechanism:	**by reference**

Optional. The kind of attribute to be obtained.

Condition Values Returned

SCA$_NORMAL	An attribute value has been successfully returned.
SCA$_NONE	Warning. An attibute-value has not been returned. There are no such attributes in the entity. This condition can be returned only if this routine is processing an entity.

Description

The SCA$QUERY_GET_ATTRI_VALUE_T routine returns the value of any attribute as a character string.

If the handle describes an attribute, then this routine returns the value of that attribute. In this case, the *attribute_kind* parameter must not be specified.

If the handle describes an entity, then this routine returns the value of the first attribute of that entity that is of the kind specified by the *attribute_kind* parameter. In this case, the *attribute_kind* parameter must be specified.

If you want to get more than one attribute value of a particular kind for an entity, you must use the routine SCA$QUERY_GET_ATTRIBUTE. This applies only to the attribute kinds SCA$K_ATTRI_NAME and SCA$K_ATTRI_ALL.

The value of any kind of attribute can be returned by this routine, except for SCA$K_ATTRI_ALL. This routine will convert to character string those attributes whose data type is not character string.

This routine does not accept the attribute-kind SCA$K_ATTRI_ALL as the value of the *attribute_kind* parameter. It is not meaningful to get just the first attribute without regard to attribute-kind.

SCA$QUERY_GET_OCCURRENCE

Gets the next occurrence in the query result that is specified as a *query_context* argument.

Format

SCA$QUERY_GET_OCCURRENCE *query_context, entity_handle*

Arguments

query_context

type:	**longword**
access:	**read/write**
mechanism:	**by reference**

An SCA query context whose occurrence are to be obtained.

entity_handle

type:	**longword**
access:	**read/write**
mechanism:	**by reference**

An SCA entity handle that describes an entity.

Condition Values Returned

SCA$_NORMAL	An occurrence has been successfully returned.
SCA$_NEWSYMBOL	An occurrence has been successfully returned. This new occurrence is of a different symbol than the occurrence that was returned by the previous call to this routine.

SCA$_NOMORE	Warning. An occurrence has not been returned. The traversal of the query result has been exhausted.

Description

The SCA$QUERY_GET_OCCURRENCE routine successively returns every occurrence in a query result. It provides one pass through all of the occurrences.

SCA$QUERY_GET_NAME

Returns the name of a query.

Format

SCA$QUERY_GET_NAME *query_context,*
query_name

Arguments

query_context

type:	**longword**
access:	**read/write**
mechanism:	**by reference**

An SCA query context whose name is to be obtained.

query_name

type:	**character string**
access:	**write only**
mechanism:	**by descriptor**

The name of the query.

Condition Value Returned

SCA$_NORMAL	The query name has been successfully returned.

Description

The SCA$QUERY_GET_NAME routine returns the name of a query.

SCA$QUERY_INITIALIZE

Initializes an SCA query context.

Format

SCA$QUERY_INITIALIZE *command_context, query_context*

Arguments

command_context

type:	**longword**
access:	**read/write**
mechanism:	**by reference**

An SCA command context.

query_context

type:	**longword**
access:	**write only**
mechanism:	**by reference**

An SCA query context to be initialized. This value is passed as an argument to other SCA query routines (SCA$QUERY_xxx).

Condition Value Returned

SCA$_NORMAL	The query context has been successfully initialized.

SCA$QUERY_INITIALIZE

Description

The SCA$QUERY_INITIALIZE routine initializes an SCA query context.

This routine must be called before any other SCA query routines.

SCA$QUERY_PARSE

Parses a query expression command string and sets up a query context if the command is syntactically correct.

Format

SCA$QUERY_PARSE *query_context, query_expression_string [,query_expression_length]*

Arguments

query_context

type:	**$SCA_QUERY_CONTEXT**
access:	**read/write**
mechanism:	**by reference**

An SCA query context that is to describe the indicated query expression.

query_expression_string

type:	**character string**
access:	**read only**
mechanism:	**by descriptor**

A query expression string.

query_expression_length

type:	**longword**
access:	**write only**
mechanism:	**by reference**

Optional. Length of the query expression, returned from the parser.

Condition Values Returned

SCA$_NORMAL	The query expression string has been successfully parsed.
SCA$_MORETEXT	Warning. The query expression string has been successfully parsed, but the text following the query expression is not a legal part of the query expression. This condition is returned only if the *query_expression_length* parameter is specified. If the *query_expression_length* parmeter is not specified, then this routine insists that the whole *query_expression_string* argument be a legal query expression; in this case all errors are signaled.

Description

The SCA$QUERY_PARSE routine parses a query expression string and sets up a query context if the command is syntactically correct.

SCA$QUERY_SELECT_OCCURRENCE

Creates a query expression that matches a specific entity.

Format

SCA$QUERY_SELECT_OCCURRENCE *query_context, entity_handle*

Arguments

query_context

type:	**longword**
access:	**read/write**
mechanism:	**by reference**

An SCA query context that is to describe a specific entity.

entity_handle

type:	**longword**
access:	**read/write**
mechanism:	**by reference**

An SCA entity handle that describes the entity that the newly defined query context is to match.

Condition Value Returned

SCA$_NORMAL	A query expression has been successfully defined.

SCA$QUERY_SELECT_OCCURRENCE

Description

The SCA$QUERY_SELECT_OCCURRENCE routine creates a query expression that matches a specific entity.

You use this routine to specify queries based on the results of previous queries. The *entity_handle* parameter is obtained by traversing the results of a previous query evaluation.

Typically, the query context of the *entity_handle* parameter is not the same as the *query_context* parameter. However, they can be the same. If they are the same query context, then that previous query is replaced with the query defined by this routine and, as a result, *entity_handle* becomes invalid.

Appendix C

Interfacing to VAXTPU Procedures

Some LSE commands depend on procedures written in the VAXTPU programming language that are present in LSE's default section file (LSE$SECTION.TPU$SECTION). These procedures must be present for LSE to function properly. For this reason, if you want to use your own VAXTPU section file, you must build it using LSE$SECTION.TPU$SECTION as a base. To do this successfully, your VAXTPU procedures must obey certain rules described in this appendix.

Digital reserves all variable names and buffer names containing the dollar sign ($) character. You must not use names containing a dollar sign ($) in your own VAXTPU code except as explained in the following sections.

C.1 VAXTPU Variables and Procedures

The following three variable names have special meaning to LSE and VAXTPU:

MESSAGE_BUFFER	The buffer to which LSE writes messages
SHOW_BUFFER	The buffer to receive output from the VAXTPU SHOW built-in
INFO_WINDOW	The window to which the VAXTPU SHOW built-in maps SHOW_BUFFER

Your section file must not redefine TPU$INITIALIZE, the VAXTPU procedure that LSE calls to set up the editing environment. LSE$SECTION.TPU$SECTION provides its own TPU$INIT_PROCEDURE. Instead, you should redefine TPU$LOCAL_INIT to perform initialization at startup time as described later in this appendix.

LSE uses the following VAXTPU variables and procedures.

TPU$LOCAL_INIT
LSE's TPU$INIT_PROCEDURE calls this procedure after it has finished LSE initialization. LSE initialization includes processing the /INITIALIZATION qualifier and reading into a buffer the input file specified on the LSEDIT command. You can supply your own TPU$LOCAL_INIT procedure to initialize your own VAXTPU variables and procedures.

LSE$CREATE_SELECT_RANGE
This procedure sets LSE$SELECT_RANGE through the following process. If LSE$START_SELECT_MARK is nonzero, it sets LSE$SELECT_RANGE to the range from LSE$START_SELECT_MARK to the current position and then zeros out LSE$START_SELECT_MARK and LSE$SELECT_IN_PROGRESS. Otherwise, if the cursor is positioned to the last string for which the user searched, it sets LSE$SELECT_RANGE to be a range containing that string. Otherwise, it sets LSE$SELECT_RANGE to zero.

LSE$SET_STATUS_LINE (window)
LSE calls this procedure whenever it wants to update the status line of a window. The procedure takes one argument—the window whose status line is to be set. If you want to change the status line for LSE, see the *VAX Text Processing Utility Manual.*

LSE$MESSAGE_WINDOW
This is a procedure that returns the window to which LSE maps MESSAGE_BUFFER.

LSE$NUMBER_OF_WINDOWS
This is a procedure that returns the number of windows mapped to the screen. Note that the number of windows may be more than two, the maximum for earlier versions of LSE.

LSE$MAIN_WINDOW
This procedure returns the top window displayed on the screen. It is compatible with earlier versions of LSE in which it was a variable that returned the window that was used in one-window mode. The current multi-window implementation based on EVE creates and deletes windows as needed, making a backwards-compatible implementation of LSE$MAIN_WINDOW impossible.

LSE$TOP_WINDOW
LSE$BOTTOM_WINDOW
These procedures return the top and bottom windows currently being displayed. They are compatible with earlier versions of LSE in which they

were variables that returned the windows that were used in two-window mode.

LSE$MAIN_BUFFER

After LSE startup, this procedure points to the buffer containing the input file that appeared on the LSEDIT command line. When you exit from LSE, LSE remembers the current cursor position in this buffer. It is compatible with earlier versions of LSE in which it was a variable. It has been replaced by the variable EVE$X_MAIN_BUFFER.

LSE$START_SELECT_MARK

This procedure returns the contents of the EVE$X_SELECT_POSITION variable. It is compatible with earlier versions of LSE in which it was a variable. The contents of EVE$SELECT_POSITION may be either the select marker set by the SET SELECT_MARK command, or a range created by the SELECT ALL command or by using the mouse. If there is no SELECT operation in progress, its value is the integer 0.

LSE$SELECT_IN_PROGRESS

This procedure returns 1 if there is a SELECT operation in progress. If no SELECT operation is in progress, then it returns 0. The value returned is computed by the TPU expression (EVE$X_SELECT_POSITION <> 0).

LSE$SELECT_RANGE

This is the range variable in which LSE$CREATE_SELECT_RANGE returns its value. LSE commands that act on the selected range use this variable.

Sample VAXTPU Procedure

The following is a VAXTPU procedure that demonstrates the use of an LSE selected range from a user-defined VAXTPU procedure. Note the use of the variables LSE$SELECT_IN_PROGRESS and LSE$SELECT_RANGE, and the procedure LSE$CREATE_SELECT_RANGE.

```
PROCEDURE sort (qual)
    ! Description:
    !   Sorts the lines in the selected range.  Complete lines should be
    !   selected.  If no qualifiers are specified, the lines in the
    !   selected range are sorted in ascending order.
    !
    ! Parameter:
    !   qual - a string beginning with "$". The remainder of the string
    !       contains qualifiers to be passed to the SORT command.  The "$" is
    !       a dummy character.  It is there to serve as a parameter when no
    !       SORT qualifiers are specified, and to prevent qualifiers for
    !       SORT from being interpreted as qualifiers on the LSE CALL
    !       command.
    !
    LOCAL sort_process,cmd,save_position,current_message;

    ! If there is a selected range, write it to a temporary file.
    IF NOT LSE$SELECT_IN_PROGRESS
    THEN
        MESSAGE ('No select active');
        RETURN;
    ENDIF;
    LSE$CREATE_SELECT_RANGE;
    WRITE_FILE (LSE$SELECT_RANGE, 'sort_input.dat');

    ! Create a subprocess in which to run SORT.  Note that terminal output
    ! from the subprocess goes to the message buffer.
    sort_process := CREATE_PROCESS (message_buffer, 'SET NOON');

    ! Build the SORT command, picking up qualifiers that were passed in.
    cmd := 'SORT/STABLE sort_input sort_output '+SUBSTR(qual,2,LENGTH(qual)-1);

    ! Display the SORT command in the message window.
    MESSAGE (cmd);

    ! Execute the SORT command in the subprocess.
    SEND (cmd, sort_process);

    ! If no messages were written to the message buffer by SORT,
    ! assume that the SORT operation succeeded and replaceed the selected range
    ! with the output from SORT.
    save_position := MARK(NONE);
    POSITION(message_buffer);
    current_message := CURRENT_LINE;
    POSITION (save_position);
    IF current_message = cmd
    THEN
        ERASE (LSE$SELECT_RANGE);
        READ_FILE ('sort_output.dat');
    ENDIF;

    ! Cleanup
    DELETE (LSE$SELECT_RANGE);
    SEND ('DELETE sort_input.dat;,sort_output.dat;',sort_process);
    DELETE (sort_process);
ENDPROCEDURE
```

To use the preceding procedure, define a SORT command as follows:

```
LSE>  DEFINE COMMAND SORT "CALL SORT $"
```

To sort the lines in the selected range in ascending order, type the following command:

```
LSE> SORT
```

To sort the lines in the selected range in descending order based on the text that begins in the 10th column and extends to, but does not include, the 20th column, type the following command:

```
LSE> SORT/KEY=(POSITION=10, SIZE=10, DESCENDING)
```

C.2 Guidelines for User-Written TPU Procedures

You can transport user-written TPU procedures from EVE to LSE. Therefore, you can use code that calls EVE procedures within LSE.

The following LSE variables are now procedures. If a TPU procedure accesses the value of the corresponding variable, but does not assign a value to it, the TPU procedure should continue to work. If a TPU procedure must change the value of one of the following variables, you should change the TPU code to use the corresponding EVE variable, if any, shown in parentheses.

- LSE$MESSAGE_WINDOW (MESSAGE_WINDOW)
- LSE$NUMBER_OF_WINDOWS (EVE$X_NUMBER_OF_WINDOWS)
- LSE$MAIN_WINDOW
- LSE$TOP_WINDOW
- LSE$BOTTOM_WINDOW
- LSE$MAIN_BUFFER (EVE$X_MAIN_BUFFER)
- LSE$START_SELECT_MARK (EVE$X_SELECT_POSITION)
- LSE$SELECT_IN_PROGRESS (returns EVE$X_SELECT_POSITION <> 0)

To tailor window status lines, see the information on EVE status-line fields in the *VAX Text Processing Utility Manual.*

C.2.1 Adding User-Written TPU Procedures

You can add user-written TPU procedures to LSE with the VAXTPU tool EVE$BUILD. You use EVE$BUILD for modifying or adding user-written TPU procedures to LSE. EVE$BUILD compiles VAXTPU code with an existing LSE section file to produce a new section file. See the *VAX Text Processing Utility Manual* for more information on using EVE$BUILD.

To extend LSE with EVE$BUILD, use the following steps:

1. Create a file called USER_MASTER.FILE that lists the files being used to extend LSE. For example:

   ```
   sys$login:abbreviation.tpu
   lseplus:auto_indent.tpu
   sys$login:customizations.tpu
   ```

2. Create a file called USER_VERSION.DAT that contains the version number to be associated with this section file, for example, V1.0.
3. Define a foreign command to use for builds. For example:

   ```
   $ BUILD == "LSEDIT/NODISP/NOINIT/COMM=SYS$EXAMPLES:EVE$BUILD"
   ```

4. Type the command that builds this module in with the existing LSE section file. You will get messages that the definitions of various EVE procedures are being superseded, which you may ignore. For example:

   ```
   $ BUILD USER
   %TPU-S-FILEIN, xxx lines read from file SYS$EXAMPLES:EVE$BUILD.TPU
   Definition of procedure EVE$BUILD_MODULE_INDENT superceded
   .
   .
   .
   Definition of procedure EVE$BUILD superceded
   Section file name [default = product name USER]:
   ```

   At this point, you must enter the name and location of the section file that you want to create.
5. Press the Return key to create a section file named USER.TPU$SECTION in the current directory.

To use this newly created section file, invoke LSE with the /SECTION qualifier and supply the full file specification that corresponds to the section file. Alternatively, you can define the logical name LSE$SECTION to be the full file specification for this new section file.

C.2.2 VAXTPU Programming with Hidden Records in LSE

With LSE, you use the COLLAPSE, EXPAND, FOCUS, and VIEW SOURCE commands for viewing source code. This code elision feature means that there can be four different types of records (or lines) in a buffer. A record may be a source record or an overview record, and either may be visible or hidden.

Source records correspond to the actual text that is read from a file, edited, and written to a file. Overview records are inserted by LSE and are representatives for source records that have been hidden or elided. Overview records themselves are hidden when the corresponding source is made visible. Overview records may also be hidden along with source records as when a set of lines containing both source lines and overview lines is collapsed to an overview.

Thus, the four types of records are as follows:

- Visible source record
- Hidden source record
- Visible overview record
- Hidden overview record

With TPU, the current position in a buffer can be on any one of these types of records. The TPU built-ins MOVE_VERTICAL and MOVE_HORIZONTAL move from record to record and are not influenced by the visibility or whether the record is an overview. A TPU procedure that does not consider visibility or overview records may not function as intended if you use the elision facility prior to calling the procedure.

After each LSE command, if the current position is not on a visible record, LSE makes the record visible. If the current position is on a hidden source record, LSE expands sufficient overviews to make the record visible. If the current position is on a hidden overview record, LSE collapses source to make the overview visible.

Overview records are not modifiable. If you attempt to alter the text, split the record, or append the record to another record, it will fail.

A number of built-ins are available for you to enhance or develop TPU procedures to work when there are overviews or hidden records in a buffer. The built-ins are listed here and described in Section C.3.

- LSE$IS_OVERVIEW
- LSE$IS_VISIBLE

- LSE$MAKE_VISIBLE
- LSE$MOVE_BY_SOURCE
- LSE$MOVE_HORIZONTAL
- LSE$MOVE_TEXT
- LSE$MOVE_VERTICAL
- LSE$NEAREST_VISIBLE
- LSE$SOURCE_ONLY

C.3 Supplemental VAXTPU Built-Ins

LSE supports new VAXTPU built-in procedures and extends some of the existing built-ins, as described in the following sections.

C.3.1 LSE$DO_COMMAND (String)

Takes a single character string as its argument. It executes the string as an LSE command. You can use this built-in to execute LSE commands from within your VAXTPU procedures.

C.3.2 LSE$GET_ENVIRONMENT(String, Keyword)

Incorporates the definitions contained in an environment file into the editing session. There are two arguments.

string

Specifies the file specification of the environment file.

keyword

Specifies the keyword (ON or OFF) that indicates whether definitions from the file should be written out by LSE's SAVE ENVIRONMENT command.

- ON—Write out the definitions.
- OFF—Do not write out the definitions.

C.3.3 GET_INFO(buffer, "language")

Returns a string representing the name of the language currently associated with the given buffer. If there is no language associated with the buffer, then the integer zero is returned.

C.3.4 GET_INFO(buffer, "overviews")

Returns either the keyword ON or the keyword OFF based on whether or not overview operations are allowed in the given buffer.

C.3.5 GET_INFO(COMMAND_LINE, item)

LSE provides the following additional COMMAND_LINE items for the GET_INFO built-in:

- CHARACTER

  Returns an integer containing the starting character position in the starting line for the edit. The first character position in the line is character 1. This is the value from the second number in the /START_POSITION qualifier, or the value from translating the logical name LSE$START_CHARACTER. This item is a synonym for the START_CHARACTER item maintained for compatibility with earlier versions of LSE.
- ENVIRONMENT

  Returns 1 if the /ENVIRONMENT qualifier is present on the command line; otherwise returns 0.
- ENVIRONMENT_FILE

  Returns a string containing the file specification from the /ENVIRONMENT qualifier. The /ENVIRONMENT qualifier specifies a list of file specifications. Each time a GET_INFO(COMMAND_LINE,"ENVIRONMENT_FILE") built-in call is done, LSE returns the next file specification in the list. It returns the null string on all calls after the end of the list is reached.

  This built-in call returns the null string if /ENVIRONMENT was not present on the command line.
- LANGUAGE

  Returns a string containing the language name from the /LANGUAGE qualifier on the command line, or the null string if /LANGUAGE was not specified.

- LINE

  Returns an integer containing the starting line number for LSE. The first line in the file is considered line 1. This is the value from the /START_POSITION qualifier or the translation of the logical name LSE$START_LINE. This item is a synonym for the START_RECORD item maintained for compatibility with earlier versions of LSE.
- SYSTEM_ENVIRONMENT

  Returns 1 if the /SYSTEM_ENVIRONMENT qualifier is present on the command line; otherwise returns 0.
- SYSTEM_ENVIRONMENT_FILE

  Returns a string containing the file specification from the /SYSTEM_ENVIRONMENT qualifier, or the null string if /SYSTEM_ENVIRONMENT is not present on the command line.
- CURRENT_FILE

  Returns 0 if the /NOCURRENT_FILE qualifier is specified on the command line, and returns 1 if the /CURRENT_FILE qualifier is specified on the command line.

C.3.6 LSE$FIND_OPEN_COMMENT (marker)

Returns a range that corresponds to the first open comment delimiter found after the marker, but on the same line as the marker.

Returns 0 if there is no language associated with the buffer containing the marker.

C.3.7 LSE$FIND_CLOSE_COMMENT (marker)

Returns a range that corresponds to the first close comment delimiter found after the marker, but on the same line as the marker.

Returns 0 if there is no language associated with the buffer, or if no close comment is found.

C.3.8 LSE$IS_OVERVIEW [(marker)]

Returns 1 if the indicated record is an overview record and 0 if it is a source record. If the maker parameter is not specified, the current record is used.

C.3.9 LSE$IS_VISIBLE [(marker)]

Returns 1 if the indicated record is a visible record and 0 if it is a hidden record. If the marker parameter is not specified, the current record is used.

C.3.10 LSE$MOVE_HORIZONTAL (integer)

Restricts the cursor to visible records. LSE does not count the characters or end-of-line on hidden lines when determining where to establish the new editing point. If the original editing point is on a hidden record, the movement to a visible record counts as a move of one line. (Similar to TPU MOVE_HORIZONTAL.)

C.3.11 LSE$MOVE_BY_SOURCE (integer)

Restricts the cursor to source records only. LSE does not count overview lines when determining where to establish the new editing point. If the original editing point is on an overview line, the movement to a source line counts as a move of one line. If the source line on to which the cursor is to move is hidden, LSE$MOVE_BY_SOURCE makes the source line visible. (Similar to the TPU MOVE_VERTICAL.)

C.3.12 LSE$MAKE_VISIBLE (marker | range)

Makes the specified records visible. If a marker is specified, LSE makes the corresponding record visible by expanding overview lines. If a range is specified, LSE makes all the source records in the range visible by expanding sufficient overviews.

C.3.13 LSE$NEAREST_VISIBLE (marker)

Moves the editing position to the beginning of the visible line nearest to the specified position. If the record at the specified position is visible, it simply moves the current editing position there. If the marker parameter is not specified, the current editing position is used. This is useful for operations that move the editing position to a new record but should not change the view, for example, moving the cursor by using a scroll bar, or moving the cursor to a window where the last position in that window has become hidden.

C.3.14 LSE$SOURCE_ONLY (range)

Returns 1 if all the source records within the range are visible; otherwise returns 0. If all the source records within the range are visible, then, as a side effect, all the hidden overview records in the range are deleted. This function is useful when writing a TPU procedure that operates on a range. It does not operate properly if there are hidden records or overview records in the range. For example, this built-in is used in the procedure that implements the FILL operation.

C.3.15 LSE$MOVE_TEXT and LSE$COPY_TEXT (string | range | buffer)

Move or copy the text from the specified string, range, or buffer to the current editing position and return the result range. If the input is a string, these functions are equivalent to MOVE_TEXT and COPY_TEXT. For ranges and buffers, the LSE functions preserve overview information. (Similar to the TPU built-ins MOVE_TEXT and COPY_TEXT.)

Overview information is language-dependent, so the language associated with the input range or buffer must be the same as the language associated with the current buffer. If the input language is not the same as the language for the current buffer, then there is a side effect, as follows:

- If the current buffer may legally accept overview records, and if the current buffer is empty, or if the current buffer has no associated language and contains no overview records, then the current buffer inherits the language of the input buffer. A buffer containing no records or only one null record is considered empty.
- In the case of LSE$COPY_TEXT, only visible records are copied. Visible overview records in the result range are marked as source records. In this case, LSE$MOVE_TEXT aborts to avoid the loss of hidden source lines.

If LSE$MOVE_TEXT is given a range, any hidden overview lines immediately preceding the range are deleted.

If there are overview records in a range or buffer, TPU functions MOVE_TEXT and COPY_TEXT change the overview records into source records. Visibility of records is preserved.

LSE$COPY_TEXT and LSE$MOVE_TEXT will not operate on an input range that includes part, but not all, of an overview line. An overview line includes the line break at its end.

C.3.16 SET (LSE$LANGUAGE, buffer, language)

Associates or disassociates a language and a buffer. See the descriptions of the SET LANGUAGE and SET NOLANGUAGE commands in the *VAX Language-Sensitive Editor and VAX Source Code Analyzer Reference Manual* for a more complete discussion of associating a language with a buffer.

The arguments to the built-in are the keyword LSE$LANGUAGE, followed by a buffer variable, followed by the language string. The literal current buffer may be used as the buffer variable. The language string may be passed as double quotes (""), which results in disassociating the language from the buffer.

C.3.17 SET (LSE$OVERVIEWS, buffer, on/off)

Enables or disables overview operations in the indicated buffer. See the descriptions of the SET OVERVIEW and SET NOOVERVIEW commands in the *VAX Language-Sensitive Editor and VAX Source Code Analyzer Reference Manual* for a more complete discussion of overview operations.

The arguments to the built-in are the keyword LSE$OVERVIEWS, followed by a buffer variable, followed by either the keyword ON or the keyword OFF. You may use the literal *current_buffer* as the buffer varible.

C.3.18 TPU Built-ins for the SCA Callable Interface

There are TPU built-ins for the SCA$QUERY_xxx functions in the new SCA callable interface. Specifically, the built-ins are as follows:

- SCA$QUERY_CLEANUP
- SCA$QUERY_COPY
- SCA$QUERY_FIND
- SCA$QUERY_GET_ATTRIBUTE
- SCA$QUERY_GET_ATTRI_KIND_T
- SCA$QUERY_GET_ATTRI_VALUE_T
- SCA$QUERY_GET_OCCURRENCE
- SCA$QUERY_GET_NAME
- SCA$QUERY_INITIALIZE

- SCA$QUERY_PARSE
- SCA$QUERY_SELECT_ENTITY

None of the other routines are available. LSE calls SCA$INITIALIZE and SCA$CLEANUP automatically on your behalf. The command context created by LSE is available in the TPU variable LSE$SCA_COMMAND_CONTEXT; you must use this as the first parameter to call SCA$QUERY_INITIALIZE.

SCA message codes are available as TPU keywords, as in the conventional format SCA$_xxx. These may be used as message constants for the TPU MESSAGE built-in.

SCA constants for the attribute kinds are available as TPU constants in the form SCA$K_ATTRI_xxx. These may be passed directly to SCA$QUERY_GET_ATTRI_VALUE_T.

LSE handles the calling sequences on your behalf. You need not be concerned with whether objects are passed by value or reference.

Note that TPU does not produce a traceback if an SCA routine signals an error.

Appendix D
Language-Specific Information

This appendix contains information of interest to VAX FORTRAN and VAX COBOL programmers. Section D.1 provides information on using VAX FORTRAN with LSE. Section D.2 provides information on using VAX COBOL with LSE.

D.1 VAX FORTRAN

Some LSE commands behave differently when the definition of the current language includes the /FORTRAN qualifier. The syntax of this qualifier is as follows:

```
/FORTRAN=[NO]ANSI_FORMAT
```

ANSI_FORMAT specifies that templates should be expanded in ANSI format. The default is NOANSI_FORMAT (tab format).

To choose a format that is different from the format specified in the /FORTRAN qualifier on the language definition, use the MODIFY LANGUAGE command and specify /FORTRAN=ANSI_FORMAT or /FORTRAN=NOANSI_FORMAT. (See the MODIFY LANGUAGE command in the Command Dictionary for more information on the /FORTRAN=[NO]ANSI_FORMAT qualifier.)

VAX FORTRAN Source Format

The VAX FORTRAN compiler supports two source-line formats: ANSI format and tab format.

VAX FORTRAN differs from the other languages supported by LSE in that each source line is divided into three fields. These fields are as follows:

- Statement number field
- Continuation field
- Statement field

In ANSI format, the first five characters contain the line number and padding blanks. The sixth character is nonblank and nonzero if the line is a continuation of the last line. The VAX FORTRAN statement field begins at the seventh character and the line terminates with the 72nd character. Any characters after the 72nd character are ignored.

In tab format, the optional line number appears first on the line and is terminated by a tab character. If the character after the tab is a nonzero digit, then that digit is the continuation field. The character after the continuation field begins the statement field. If the character after the tab is a nondigit character, then that character begins the statement field.

Token and Placeholder Definitions

The bodies of VAX FORTRAN tokens and placeholders should be entered as legal source lines in tab format. This allows LSE to determine the fields and permit the lines to contain statement number and continuation fields. If ANSI_FORMAT is specified, LSE converts the body to ANSI format when the body is expanded into the source file. A placeholder appearing in the statement number field is limited to five characters and must be a terminal placeholder.

Entering and Erasing Text

When a placeholder is erased from the statement number field in ANSI_FORMAT mode, it is replaced with blanks.

When a placeholder in the statement field is expanded, the statement number and continuation fields of the first line of the placeholder body are ignored and the statement field is inserted at the position vacated by the placeholder.

Note that the procedure for expanding tokens is identical.

Indentation

Indentation of FORTRAN statements is done only in the statement field, rather than at the beginning of a line as for other languages. Tab stops are set for the statement field only, with column 1 being the first character of the statement field.

An ENTER TAB or TAB command (bound to the TAB key) issued at the beginning of a line inserts a tab character in NOANSI mode. In ANSI mode, a tab at the beginning of a line moves the cursor to the statement field. Erasing one character backwards at that point puts the cursor in the continuation field.

An ENTER TAB or TAB command, entered at the beginning of the statement field, inserts the current indentation.

D.2 VAX COBOL

This section provides a description of the convention used for placeholder definitions within VAX COBOL.

Notation

Within the VAX COBOL placeholders, you will see the following three notations:

- [placeholder ..]
- [placeholder]...
- [placeholder]....

In the notation [placeholder ..], the space and two dots following the placeholder indicate that, upon expansion, you will see more details of the placeholder and not just the keywords that appear within the brackets.

The notation, [placeholder]..., is a list placeholder. The three dots indicate that the placeholder will be duplicated upon expansion.

The notation, [placeholder]...., is a list placeholder followed by punctuation, in this case, a period. The first three dots indicate that the placeholder will be duplicated upon expansion. Upon removal of the duplicated placeholder, a period will end the line.

Appendix E
Packages

LSE provides a mechanism for defining your own packages. The package facility includes two VAXTPU sets of procedures to help you write your own packages.

E.1 VAXTPU Procedures for the Package Facility

The VAXTPU procedures generate the appropriate DEFINE TOKEN and DEFINE PLACEHOLDER commands for each routine or parameter. The VAXTPU procedures, indicated by the /ROUTINE_EXPAND qualifier, generate a token definition for the routine name. The VAXTPU procedures, indicated by the /PARAMETER_EXPAND qualifier, generate one or more placeholder definitions for each parameter name.

LSE comes with two sets of predefined VAXTPU procedures that you can use to perform these expansions. For each language associated with the package, there must be a ROUTINE_EXPAND procedure and a PARAMETER_EXPAND procedure. You can specify these procedures with the /ROUTINE_EXPAND and /PARAMETER_EXPAND qualifiers. The value that you specify for these qualifiers in the DEFINE PACKAGE command must be a prefix shared by all the corresponding procedures. LSE determines the actual procedure name by concatenating the prefix value and the appropriate language name.

The procedures supplied with LSE are as follows:

- LSE$PKG_EXPAND_ROUT_ADA
- LSE$PKG_EXPAND_PARM_ADA
- LSE$PKG_EXPAND_ROUT_BASIC
- LSE$PKG_EXPAND_PARM_BASIC
- LSE$PKG_EXPAND_ROUT_BLISS

- LSE$PKG_EXPAND_PARM_BLISS
- LSE$PKG_EXPAND_ROUT_C
- LSE$PKG_EXPAND_PARM_C
- LSE$PKG_EXPAND_ROUT_COBOL
- LSE$PKG_EXPAND_PARM_COBOL
- LSE$PKG_EXPAND_ROUT_FORTRAN
- LSE$PKG_EXPAND_PARM_FORTRAN
- LSE$PKG_EXPAND_ROUT_PASCAL
- LSE$PKG_EXPAND_PARM_PASCAL
- LSE$PKG_EXPAND_ROUT_PLI
- LSE$PKG_EXPAND_PARM_PLI

You use these routines by specifying /ROUTINE_EXPAND = LSE$PKG_EXPAND_ROUT_ and /PARAMETER_EXPAND = LSE$PKG_EXPAND_PARM_.

If you want to write your own VAXTPU procedures for these purposes, they must conform to the following restrictions:

- When LSE needs to generate a token definition from a routine definition, it calls the VAXTPU procedure specified by the /ROUTINE_EXPAND qualifier. A typical VAXTPU procedure for the /ROUTINE_EXPAND qualifier appears as follows:

```
PROCEDURE my_routine_expand_somelanguage
LOCAL command_string, {other tpu local variables}...;

 . . .

command_string :=     'DEFINE TOKEN '
                    + routine_name
                    + '/LANGUAGE = somelanguage '
                    + <any other qualifiers for the DEFINE TOKEN command>;

LSE$DO_COMMAND (command_string) ;

< tpu code to generate the body of the token, using LSE$DO_COMMAND>

LSE$DO_COMMAND ('END DEFINE');

ENDPROCEDURE;
```

All of the other information that is included in the DEFINE ROUTINE command is passed to your VAXTPU routine by means of the VAXTPU global variables. The following global variables are defined:

```
LSE$PKG_ROUT_NAME   contains the name of the routine being defined,
                        enclosed in quotes (")
LSE$PKG_ROUT_LANG   contains the name of the language
LSE$PKG_ROUT_DESC   contains the value of the /DESCRIPTION parameter
LSE$PKG_ROUT_TOP    contains the value of the /TOPIC parameter
LSE$PKG_ROUT_PACK   contains the package name
LSE$PKG_ROUT_PARM   contains the list of parameters associated with the
                        routine being defined.  Each parameter is enclosed in
                        quotes and separated from the next by a carriage return,
                        line feed pair; that is, the tpu string
                        ASCII(13) + ASCII(10)
LSE$PKG_ROUT_OPT    contains a list of flags, in one-to-one correspondence
                        to the list of parameters.  Each flag can be either
                        an O, indicating that the parameter is optional,
                        or an R, indicating that the parameter is
                        required.  Each flag is separated from the next by
                        a single-space character.
LSE$PKG_ROUT_MECH   contains a list of flags, in one-to-one correspondence
                        to the list of parameters.  Each flag can be either
                        a V, indicating BY VALUE, an R, indicating
                        BY REFERENCE, a D, indicating BY DESCRIPTOR, or
                        a U, indicating UNKNOWN.  Each flag is separated
                        from the next by a single-space character.
```

- When LSE needs to generate a placeholder definition from a parameter definition, it calls the VAXTPU procedure specified by the /PARAMETER_EXPAND qualifier of the package. A typical VAXTPU procedure for /PARAMETER_EXPAND appears as follows:

```
PROCEDURE my_parameter_expand_somelanguage
LOCAL command_string, {other tpu local variables}...;

 . . .

command_string :=     'DEFINE PLACEHOLDER '
                    + routine_name
                    + '/LANGUAGE = somelanguage '
                    + '/TYPE = TERMINAL/SEPARATOR = ","'
                    + {other qualifiers for the DEFINE PLACEHOLDER command};

LSE$DO_COMMAND (command_string) ;

 . . .

ENDPROCEDURE;
```

The following global variables are defined for use by the VAXTPU procedure specified by the /PARAMETER_EXPAND qualifier:

```
LSE$PKG_PARA_NAME   contains the name of the parameter to be defined
LSE$PKG_PARA_LANG   contains the name of the language
```

E.2 Example Procedures

This section presents the TPU expansion procedures for Pascal and some of the support routines. The first TPU procedure, LSE$PKG_EXPAND_ROUT_PASCAL, defines a token for a package routine. It calls other TPU procedures that you can use as is or redefine according to your needs.

The second procedure, LSE$PKG_EXPAND_PARM_PASCAL, defines two placeholders for each parameter. Since the Pascal system service routines are in a keyword format (for example, %[p1 := %{p1}%]%), a placeholder must be defined for p1 and p1 := %{p1}%. The first placeholder is defined in the procedure LSE$PKG_DEFINE_PARAMETER, and the second in LSE$PKG_EXPAND_PARAM_PASCAL.

The following called procedures are also listed:

- LSE$PKG_PAD_NAME
- LSE$PKG_DEFINE_TOKEN
- LSE$PKG_GET_PARAM
- LSE$PKG_DEFINE_PARAMETER

Note that the TPU built-in procedure, change_case, is called to force the case of expansions. You can modify the expansion routines to use CHANGE_CASE to follow any case convention you want.

```
PROCEDURE lse$pkg_expand_rout_pascal
!++
! FUNCTIONAL DESCRIPTION:
!
!   This routine generates a Pascal token definition from a parameter
!   definition, using keyword syntax.
!
! FORMAL PARAMETERS:
!
!   None
!
```

```
! IMPLICIT INPUTS:
!
!   LSE$PKG_ROUT_NAME
!       The name of the routine to be defined.
!
!   LSE$PKG_ROUT_PARM
!       The list of parameters of the routine, separated by spaces.
!
!   LSE$PKG_ROUT_OPT
!       A list of flags, in one-to-one correspondence with the list of
!       parameters.  Each flag can be either O, indicating optional, or
!       R, indicating required.  Each flag is separated from the next by a
!       space.
!
! IMPLICIT OUTPUTS:
!
!   None
!
! ROUTINE VALUE:
!
!   None
!
! SIDE EFFECTS:
!
!   A token definition is issued.
!--
    LOCAL
        proc_name,
        command_string,
        cur_param,
        cur_option,
        param_name,
        keyword_param,
        mech;

    ! Start the DEFINE TOKEN command.
    lse$pkg_define_token;

    ! Remove quotes from procedure name.
    proc_name := SUBSTR(LSE$PKG_ROUT_NAME, 2, LENGTH(LSE$PKG_ROUT_NAME)-2);
    ! Format the call with the procedure name in lowercase.
    command_string := '"' + proc_name;
    CHANGE_CASE(command_string,LOWER);

    IF LSE$PKG_ROUT_PARM = '' THEN
        ! The call consists of just the procedure name
        command_string := command_string + '"';
        LSE$DO_COMMAND(command_string);
    ELSE
        ! The call has parameters

        ! Form the first line of the call.
        !   First line is just the procedure name and open parenthesis.
        command_string := command_string + ' (';
        LSE$DO_COMMAND(command_string);

        ! Move a required parameter to the beginning of the list.
        !   This avoids a problem in erasing a comma after first
        !   parameter if it is optional.
        lse$pkg_reorder_params (LSE$PKG_ROUT_PARM, LSE$PKG_ROUT_OPT);
    ! Loop for each parameter.
    LOOP
        EXITIF lse$pkg_get_param (cur_param,
                                  cur_option,
                                  LSE$PKG_ROUT_PARM,
                                  LSE$PKG_ROUT_OPT) = 0;

        ! Remove passing mechanism .x suffix (x = v, d, or r).
        keyword_param := lse$pkg_remove_mech (cur_param, mech);

        ! Modify parameter names that conflict with Pascal keywords.
        IF keyword_param = "TYPE"
        THEN
            keyword_param := keyword_param + '_';
        ENDIF;

        lse$pkg_pad_name (keyword_param, param_name);

        ! Form the template line for the parameter.
        command_string := '"' + ASCII(9);
        IF cur_option = "O"
        THEN
            ! optional parameter
            command_string := command_string
                + '%[' + param_name + ' := %{' + cur_param + '}%]%'
        ELSE
            ! required parameter
            command_string := command_string
                + param_name + ' := %{' + cur_param + '}%'
            ENDIF;
        IF LSE$PKG_ROUT_PARM = '' THEN      ! No more parameters
            ! Complete the call statement.
            command_string := command_string + ')';
        ELSE
            ! Add a separator after the parameter.
            command_string := command_string + ',';
        ENDIF ;
        ! Make the line lowercase.
        CHANGE_CASE (command_string,LOWER);

        ! Add the line to the token definition.
        LSE$DO_COMMAND (command_string);

    ENDLOOP;
ENDIF; ! parameter string is/isn't empty

! End the DEFINE TOKEN command
LSE$DO_COMMAND ("end define") ;

ENDPROCEDURE
```

```
PROCEDURE lse$pkg_expand_parm_pascal
!++
! FUNCTIONAL DESCRIPTION:
!
!   This routine generates Pascal placeholder definitions from a parameter
!   definition, for keyword syntax.
!
! FORMAL PARAMETERS:
!
!   None
!
! IMPLICIT INPUTS:
!
!   LSE$PKG_PARA_NAME
!       The name of the placeholder to define.
!
! IMPLICIT OUTPUTS:
!
!   None
!
! ROUTINE VALUE:
!
!   None
!
! SIDE EFFECTS:
!
!   Two placeholder definitions are issued.
!--
    LOCAL
        command_string,
        name_noquote,
        padded_key,
        keyword_name,
        mech;

    ! Define a placeholder for the parameter.
    lse$pkg_define_parameter('');

    ! Define a placeholder of the form "name := %{name}%".
    !    This is done in case the parameter is optional.

    ! Strip the quotes off the name
    name_noquote := SUBSTR( LSE$PKG_PARA_NAME,
                            2,
                            LENGTH(LSE$PKG_PARA_NAME) - 2) ;

    ! Remove passing mechanism .x suffix (x = v, d, or r).
    keyword_name := lse$pkg_remove_mech (name_noquote, mech);

    ! Modify parameter names that conflict with Pascal keywords
    IF keyword_name = 'TYPE'
    THEN
        keyword_name := keyword_name + '_';
    ENDIF;
    lse$pkg_pad_name (keyword_name, padded_key) ;

    ! Do the DEFINE PLACEHOLDER command.
    command_string :=
        'define placeholder /language=pascal /separator="," "' +
                        padded_key + ' := %{' + name_noquote + '}%"' ;
    CHANGE_CASE (command_string, lower);
    LSE$DO_COMMAND (command_string) ;
    ! Do the body.
    command_string := '"' + padded_key + ' := %{' + name_noquote + '}%"' ;
    CHANGE_CASE (command_string, LOWER);
    LSE$DO_COMMAND (command_string) ;

    ! End the definition.
    LSE$DO_COMMAND ('end define') ;
ENDPROCEDURE

PROCEDURE lse$pkg_pad_name (cur_param, p_keyword)
!++
! FUNCTIONAL DESCRIPTION:
!
!   Pads a parameter name so that it is at least six characters long.  This
!   is for use by keyword-style routine calls, so that the intermediate
!   assignment operations, which separate the keyword from the parameter value,
!   line up properly.
!
! FORMAL PARAMETERS:
!
!   cur_param
!       The parameter name to be padded.
!
!   p_keyword
!       The result of padding the parameter name.
!
! IMPLICIT INPUTS:
!
!   None
!
! IMPLICIT OUTPUTS:
!
!   None
!
! ROUTINE VALUE:
!
!   None
!
! SIDE EFFECTS:
!
!   p_keyword is set as indicated above
!--

    LOCAL
        len,            !* the length of cur_param
        i ;

    p_keyword := cur_param;
    ! Pad the p_keyword so it's six letters long.
    !   This tends to make the keyword calls to system services look
    !   better.
    len := LENGTH (p_keyword) ;
    IF len < 6 THEN
        i := 0 ;
        LOOP EXITIF i = 6 - len ;
            p_keyword := p_keyword + " " ;
            i := i + 1 ;
        ENDLOOP ;
    ENDIF ;
ENDPROCEDURE

PROCEDURE lse$pkg_define_token
!++
! FUNCTIONAL DESCRIPTION:
!
!   This routine generates LSE DEFINE TOKEN commands for routines.
!   It issues only the DEFINE TOKEN token-name, with qualifiers, and
!   leaves the editor in a state ready to process the definition of
!   the body of the token.  This procedure is suitable for being
!   called from any procedure that needs to define a token from a
!   routine definition; the calling procedure is responsible for
!   defining the body of the routine and issuing the closing END
!   DEFINE command.
!
! FORMAL PARAMETERS:
!
!   None
!
! IMPLICIT INPUTS:
!
!   LSE$PKG_ROUT_NAME
!       The name of the routine to be defined.
!
!   LSE$PKG_ROUT_LANG
!       The name of the language for which to define the routine.
!
!   LSE$PKG_ROUT_DESC
!       The description string for the routine.
!
!   LSE$PKG_ROUT_TOP
!       The topic string for the routine.
!
! IMPLICIT OUTPUTS:
!
!   None
!
! ROUTINE VALUE:
!
!   None
!
! SIDE EFFECTS:
!
!   Begins a DEFINE TOKEN definition.  The next calls to
!   LSE$DO_COMMAND must complete the definition.
!
```

```
! MODIFICATION HISTORY:
!
!--
    LOCAL
        proc_name,       ! name of routine being defined, with quotes removed
        command_string;  ! command string to send to LSE$DO_COMMAND

    ! Form DEFINE TOKEN command string
    command_string :=
        'define token ' + LSE$PKG_ROUT_NAME
            + ' /language = ' + LSE$PKG_ROUT_LANG
            + ' /description = "' + LSE$PKG_ROUT_DESC
            + '" /topic_string = "' + LSE$PKG_ROUT_TOP + '"';

    ! Execute the DEFINE TOKEN command
    LSE$DO_COMMAND (command_string) ;

ENDPROCEDURE;

PROCEDURE lse$pkg_get_param (param, option, param_line, option_line)
!++
! FUNCTIONAL DESCRIPTION:
!
!   Return the first parameter and option from the given parameter
!   lists and option line, removing them from the lists.
!
! FORMAL PARAMETERS:
!
!   param
!       On exit, this will be the first parameter from the param line.
!
!   option
!       On exit, this will be the first option field from the option line.
!
!   param_line
!       A list of parameters for a routine, as in LSE$PKG_ROUT_PARM.  On exit,
!       the first parameter from the list will have been removed.
!
!   option_line
!       A list of option flags for a routine's parameter list, as in
!       LSE$PKG_ROUT_OPT.  On exit, the first option from the list will have
!       been removed.
!
! IMPLICIT INPUTS:
!
!   None
!
! IMPLICIT OUTPUTS:
!
!   None
!
! ROUTINE VALUE:
!
!   0 - if there were no more parameters
!   1 - if a parameter name is returned
!
! SIDE EFFECTS:
!
!   param_line and option_line are changed as indicated above
!--
```

```
    LOCAL
        blank_idx ;      ! ** location of blanks in parameter lines

    ! Locate a parameter in param_line.
    blank_idx := INDEX (param_line, ASCII(13)+ASCII(10) );

    ! Return if no more parameters.
    IF blank_idx <= 1 THEN
        param_line := '' ;
        RETURN (0) ;
    ENDIF ;

    ! Get parameter, stripping off the outside set of quotes.
    param := SUBSTR (param_line, 2, blank_idx - 3) ;

    ! Remove parameter from param_line.
    param_line := SUBSTR (param_line, blank_idx + 2, LENGTH(param_line) ) ;

    ! Get option and remove from option_line.
    option := SUBSTR (option_line, 1, 1) ;
    option_line := SUBSTR (option_line, 3, LENGTH(option_line) ) ;
    RETURN (1) ;
ENDPROCEDURE

PROCEDURE lse$pkg_define_parameter(qualifiers)
!++
! FUNCTIONAL DESCRIPTION:
!
!   This procedure issues a standard DEFINE PARAMETER command for the parameter
! currently being expanded.  This routine is suitable for being called from
! any procedure that needs to define a placeholder from a parameter.  Note that
! unlike lse$pkg_define_token, this routine generates a complete placeholder
! definition.
!
! FORMAL PARAMETERS:
!
!   qualifiers
!       A string containing any additional qualifiers to be added to the
!       placeholder definition.  Most commonly, this will be either empty
!       or just a separator definition (e.g., '/SEPARATOR=","').  Note that
!       the parameter must be a complete qualifier or sequence of qualifier,
!       in legal LSE syntax.  Furthermore, since this routine automatically
!       adds a /type=terminal and a /language=lse$pkg_para_lang to the
!       placeholder definitions, these two qualifiers may NOT be included in
!       the qualifiers parameters.
!
!
! IMPLICIT INPUTS:
!
!   LSE$PKG_PARA_NAME
!       The name of the placeholder to define.
!
!   LSE$PKG_PARA_LANG
!       The language for which to define the placeholder.
!
! IMPLICIT OUTPUTS:
!
!   None
!
```

```
! ROUTINE VALUE:
!
!   None
!

! SIDE EFFECTS:
!
!   A new placeholder is defined.
!--
    LOCAL
        command_string,
        name_noquote,
        mech;

    ! Form DEFINE PLACEHOLDER command string
    command_string :=
        'define placeholder ' + LSE$PKG_PARA_NAME +
        ' /type=terminal /language=' + LSE$PKG_PARA_LANG
        + qualifiers;

    ! Force to lowercase
    CHANGE_CASE (command_string, LOWER);

    ! Execute the DEFINE PLACEHOLDER command
    LSE$DO_COMMAND (command_string);

    ! Strip the quotes off the name
    name_noquote := SUBSTR(LSE$PKG_PARA_NAME, 2, LENGTH(LSE$PKG_PARA_NAME) - 2);

    ! Remove passing mechanism .x suffix (x = v, d, or r).
    name_noquote := lse$pkg_remove_mech (name_noquote, mech);

    ! Do the body line.
    command_string := '"The actual data you want to pass to parameter ' +
                name_noquote + '."';
    LSE$DO_COMMAND (command_string);

    ! Do a body line for the passing mechanism.
    IF mech = 'V' THEN
        LSE$DO_COMMAND('"The parameter is passed by value."');
    ELSE IF mech = 'R' THEN
        LSE$DO_COMMAND('"The parameter is passed by reference."');
    ELSE IF mech = 'D' THEN
        LSE$DO_COMMAND('"The parameter is passed by descriptor."');
    ENDIF; ENDIF; ENDIF;
    ! End the DEFINE PLACEHOLDER command
    LSE$DO_COMMAND ("end define")
ENDPROCEDURE

PROCEDURE lse$pkg_remove_mech(param_name, mech_char)
!++
! FUNCTIONAL DESCRIPTION:
!
!   This procedure removes a suffix from a parameter name of the form
!   name.suffix.  The suffix must be either v, d, or r and
!   indicates that the parameter is passed by value, descriptor, or
!   reference, respectively.
!
! FORMAL PARAMETERS:
!
!   param_name
!       The name of the parameter.
!
!   mech_char
!       Set to the suffix character removed from param_name (uppercase).
!
! IMPLICIT INPUTS:
!
!   None
!
! IMPLICIT OUTPUTS:
!
!   None
!
! ROUTINE VALUE:
!
!   The parameter name without the .suffix.
!
! SIDE EFFECTS:
!
!   None
!--
    LOCAL
        param_length,
        mech_suffix,
        mech_separator;

    mech_char := '';
    param_length := LENGTH (param_name) ;
    IF param_length < 2 THEN RETURN (param_name) ENDIF;
    ! Get last character from param_name.
    mech_suffix := SUBSTR(param_name, param_length, 1);
    ! Get second-to-last character from param_name.
    mech_separator := SUBSTR(param_name, param_length - 1, 1);
    CHANGE_CASE (mech_suffix, UPPER);
    IF ((mech_suffix = 'V') OR (mech_suffix = 'D') OR (mech_suffix = 'R'))
            AND (mech_separator = '.') THEN
        mech_char := mech_suffix;
        RETURN ( SUBSTR (param_name, 1, param_length - 2) ) ;
    ENDIF;

    RETURN (param_name);
ENDPROCEDURE
```

Appendix F

LSE and EVE Commands

Table F–1 lists the EVE commands with the corresponding LSE commands.

Table F–1: Corresponding EVE and LSE Commands

EVE Command	LSE Command
@	None
ATTACH	ATTACH
BOTTOM	GOTO BOTTOM
BUFFER	GOTO BUFFER
CAPITALIZE WORD	CAPITALIZE WORD
CENTER LINE	CENTER LINE
CHANGE DIRECTION	CHANGE DIRECTION
CHANGE MODE	CHANGE TEXT_ENTRY_MODE
DCL	DCL
DEFINE KEY	DEFINE KEY
DELETE BUFFER	DELETE BUFFER
DELETE WINDOW	DELETE WINDOW
DO	GOTO COMMAND
END OF LINE	GOTO LINE/BOUND/FORWARD
ENLARGE WINDOW	ENLARGE WINDOW
ERASE CHARACTER	ERASE/TO CHARACTER/REVERSE
ERASE LINE	ERASE/TO LINE/BEGINNING/FORWARD

(continued on next page)

Table F–1 (Cont.): Corresponding EVE and LSE Commands

EVE Command	LSE Command
ERASE PREVIOUS WORD	ERASE WORD/PREVIOUS
ERASE START OF LINE	ERASE/TO LINE/BEGINNING/REVERSE
ERASE WORD	ERASE WORD/NEXT
EXIT	EXIT
EXTEND ALL	EXTEND *
EXTEND EVE	EXTEND
EXTEND THIS	EXTEND /INDICATED
EXTEND TPU	DO /TPU
FILL	FILL
FILL PARAGRAPH	FILL
FILL RANGE	FILL
FIND	SEARCH
FORWARD	SET FORWARD
GET FILE	GOTO FILE
GOTO	GOTO MARK
HELP	HELP
INCLUDE FILE	READ
INSERT HERE	PASTE
INSERT MODE	SET INSERT
INSERT PAGE BREAK	None
LEARN	DEFINE KEY/LEARN
LINE	LINE
LOWERCASE WORD	LOWERCASE WORD
MARK	SET MARK
MOVE BY LINE	GOTO LINE/BREAK
MOVE BY PAGE	GOTO PAGE
MOVE BY WORD	GOTO WORD/BEGINNING/CURRENT
MOVE DOWN	GOTO CHARACTER/VERTICALLY/FORWARD
MOVE LEFT	GOTO CHARACTER/HORIZONTALLY/REVERSE

(continued on next page)

Table F–1 (Cont.): Corresponding EVE and LSE Commands

EVE Command	LSE Command
MOVE RIGHT	GOTO CHARACTER/HORIZONTALLY/FORWARD
MOVE UP	GOTO CHARACTER/VERTICALLY/REVERSE
NEXT SCREEN	GOTO SCREEN/FORWARD
NEXT WINDOW	NEXT WINDOW
ONE WINDOW	ONE WINDOW
OTHER WINDOW	NEXT WINDOW
OVERSTRIKE MODE	SET OVERSTRIKE
PREVIOUS SCREEN	GOTO SCREEN/REVERSE
PREVIOUS WINDOW	PREVIOUS WINDOW
QUIT	QUIT
QUOTE	QUOTE
RECALL	RECALL
REFRESH	REFRESH
REMEMBER	END DEFINE
REMOVE	CUT
REPEAT	REPEAT
REPLACE	SUBSTITUTE
RESET	None
RESTORE	UNERASE
RESTORE CHARACTER	UNERASE CHARACTER
RESTORE LINE	UNERASE LINE
RESTORE SENTENCE	None
RESTORE WORD	UNERASE WORD
RETURN	ENTER LINE
REVERSE	SET REVERSE
SAVE EXTENDED EVE	SAVE SECTION
SAVE EXTENDED TPU	SAVE SECTION
SELECT	TOGGLE SELECT_MARK
SET CURSOR BOUND	SET CURSOR BOUND

(continued on next page)

Table F–1 (Cont.): Corresponding EVE and LSE Commands

EVE Command	LSE Command
SET CURSOR FREE	SET CURSOR FREE
SET FIND NOWHITESPACE	SET SEARCH NOSPAN_SPACE
SET FIND WHITESPACE	SET SEARCH SPAN_SPACE
SET LEFT MARGIN	SET LEFT MARGIN
SET NOPENDING DELETE	SET MODE NOPENDING_DELETE
SET NOWRAP	SET NOWRAP
SET PENDING DELETE	SET MODE PENDING_DELETE
SET RIGHT MARGIN	SET RIGHT MARGIN
SET SCROLL MARGINS	SET SCROLL_MARGINS
SET SHIFT KEY	DEFINE KEY /STATE=GOLD
SET TABS AT	None
SET TABS EVERY	None
SET TABS INVISIBLE	SET MODE TABS=INVISIBLE
SET TABS VISIBLE	SET MODE TABS=VISIBLE
SET WIDTH	SET SCREEN WIDTH
SET WILDCARD ULTRIX	SET SEARCH PATTERN=ULTRIX
SET WILDCARD VMS	SET SEARCH PATTERN=VMS
SET WRAP	SET WRAP
SHIFT LEFT	SHIFT/REVERSE
SHIFT RIGHT	SHIFT/FORWARD
SHOW	SHOW BUFFER
SHOW KEY	SHOW KEY
SHOW SUMMARY	SHOW SUMMARY
SHOW SYSTEM BUFFERS	SHOW BUFFERS/SYSTEM_BUFFERS
SHOW WILDCARD	None
SHRINK WINDOW	SHRINK WINDOW
SPACE	ENTER SPACE
SPAWN	SPAWN
SPELL	SPELL

(continued on next page)

Table F–1 (Cont.): Corresponding EVE and LSE Commands

EVE Command	LSE Command
SPLIT WINDOW	SPLIT WINDOW
START OF LINE	GOTO LINE/BOUND/REVERSE
STORE TEXT	None
TAB	TAB
TOP	GOTO TOP
TPU	DO/TPU "string"
TWO WINDOWS	TWO WINDOWS
UNDEFINE KEY	DELETE KEY
UPPERCASE WORD	UPPERCASE WORD
WHAT LINE	WHAT LINE
WILDCARD FIND	None
WRITE FILE	WRITE

Appendix G

Providing 7-Bit Terminal Support for Code Elision

You can use the VAX Terminal Fallback Facility (TFF) to resolve the problem of VT100 terminals displaying unrecognizable characters in place of the double-angle brackets («») displayed on VT200 terminals. The Terminal Fallback Facility translates the double-angle brackets to single-angle brackets. Have your system manager use the following procedure:

1. Enable TFF by including the following commands in the system startup procedure SYS$MANAGER:SYSTARTUP_V5.COM:

   ```
   $ @SYS$MANAGER:TFF$STARTUP.COM
   ```

2. Add the commands to load the default system fallback and compose sequence tables to the file SYS$MANAGER:TFF$STARTUP.COM.

   For example, to load the necessary fallback and compose sequence table for use in North America, the system manager would add the following commands:

   ```
   $ RUN SYS$SYSTEM:TFU
   SET LIBRARY SYS$SYSTEM:TFF$MASTER  ! Define the library of tables
   LOAD TABLE ASCII_OVST              ! Load for hardcopy ASCII terminal
   SET DEFAULT_TABLE ASCII            ! Set default to ASCII
   EXIT
   $ EXIT
   ```

Once this has been done, you can use the fallback utility. To enable terminal fallback, type the following command:

```
$ SET TERMINAL/FALLBACK
```

From this point forward terminal fallback is enabled. If you want to disable terminal fallback, type the following command:

```
$ SET TERMINAL/NOFALLBACK
```

If the SOFT_COMPOSE feature is enabled, you must rebind the ERASE PLACEHOLDER and UNERASE PLACEHOLDER keys to something other than CTRL/K. This is because CTRL/K is reserved by TFF to signal the initiation of a compose sequence. You may want to use the CTRL/space and GOLD-CTRL/space key bindings, but this produces an ASCII NULL, which may cause problems with some communications equipment. It is recommended that SOFT_COMPOSE be disabled unless it is required.

Index

How to Order Additional Documentation

Technical Support

If you need help deciding which documentation best meets your needs, call 800-343-4040 before placing your electronic, telephone, or direct mail order.

Electronic Orders

To place an order at the Electronic Store, dial 800-DEC-DEMO (800-332-3366) using a 1200- or 2400-baud modem. If you need assistance using the Electronic Store, call 800-DIGITAL (800-344-4825).

Telephone and Direct Mail Orders

Your Location	Call	Contact
Continental USA, Alaska, or Hawaii	800-DIGITAL	Digital Equipment Corporation P.O. Box CS2008 Nashua, New Hampshire 03061
Puerto Rico	809-754-7575	Local Digital subsidiary
Canada	800-267-6215	Digital Equipment of Canada Attn: DECdirect Operations KAO2/2 P.O. Box 13000 100 Herzberg Road Kanata, Ontario, Canada K2K 2A6
International	————	Local Digital subsidiary or approved distributor
Internal[1]	————	USASSB Order Processing - WMO/E15 *or* U.S. Area Software Supply Business Digital Equipment Corporation Westminster, Massachusetts 01473

[1]For internal orders, you must submit an Internal Software Order Form (EN-01740-07).

Reader's Comments

VAX Language-Sensitive Editor and VAX Source Code Analyzer Reference Manual
AA–PAJKA–TK

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

I would like to see more/less ______________________________

____

What I like best about this manual is ______________________________

____

What I like least about this manual is ______________________________

____

I found the following errors in this manual:

Page Description

______ ______________________________

______ ______________________________

______ ______________________________

Additional comments or suggestions to improve this manual:

____

____

____

I am using **Version** ______ of the software this manual describes.

Name/Title ______________________ Dept. __________

Company ______________________ Date __________

Mailing Address ______________________________

______________________ Phone __________

— Do Not Tear - Fold Here and Tape

digital™

No Postage
Necessary
if Mailed
in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01-3/J35
110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

— Do Not Tear - Fold Here

Reader's Comments

VAX Language-Sensitive Editor and VAX Source Code Analyzer Reference Manual
AA–PAJKA–TK

Please use this postage-paid form to comment on this manual. If you require a written reply to a software problem and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Thank you for your assistance.

I rate this manual's:	Excellent	Good	Fair	Poor
Accuracy (software works as manual says)	☐	☐	☐	☐
Completeness (enough information)	☐	☐	☐	☐
Clarity (easy to understand)	☐	☐	☐	☐
Organization (structure of subject matter)	☐	☐	☐	☐
Figures (useful)	☐	☐	☐	☐
Examples (useful)	☐	☐	☐	☐
Index (ability to find topic)	☐	☐	☐	☐
Page layout (easy to find information)	☐	☐	☐	☐

I would like to see more/less ______________________________

____

What I like best about this manual is ______________________________

____

What I like least about this manual is ______________________________

____

I found the following errors in this manual:

Page	Description
______	______________________________
______	______________________________
______	______________________________

Additional comments or suggestions to improve this manual:

____

____

____

I am using **Version** ______ of the software this manual describes.

Name/Title ______________________ Dept. ____________

Company ______________________ Date ____________

Mailing Address ______________________________

______________________ Phone ____________

— Do Not Tear - Fold Here and Tape

digital™

No Postage
Necessary
if Mailed
in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

DIGITAL EQUIPMENT CORPORATION
Corporate User Publications—Spit Brook
ZK01-3/J35
110 SPIT BROOK ROAD
NASHUA, NH 03062-9987

— Do Not Tear - Fold Here